DAHLGREN DIVISION
NAVAL SURFACE WARFARE CENTER
Silver Spring, Maryland 20903-5640

NSWCDD/MP-94/122

# PROCEEDINGS OF THE 1994 COMPLEX SYSTEMS ENGINEERING SYNTHESIS AND ASSESSMENT TECHNOLOGY WORKSHOP (CSESAW '94), 19-20 JULY 1994



STEVEN HOWELL, COORDINATOR

SYSTEMS RESEARCH AND TECHNOLOGY DEPARTMENT

12 JANUARY 1995

NSWCDD/MP-94/122

# PROCEEDINGS OF THE 1994 COMPLEX SYSTEMS ENGINEERING SYNTHESIS AND ASSESSMENT TECHNOLOGY WORKSHOP (CSESAW '94), 19-20 JULY 1994

STEVEN HOWELL, COORDINATOR
SYSTEMS RESEARCH AND TECHNOLOGY DEPARTMENT



12 JANUARY 1995



NAVAL SURFACE WARFARE CENTER
DAHLGREN DIVISION • WHITE OAK DETACHMENT
Silver Spring, Maryland 20903-5640

## *FOREWORD*

1994 COMPLEX SYSTEMS ENGINEERING SYNTHESIS AND ASSESSMENT TECHNOLOGY WORKSHOP (CSESAW '94)

This is the fourth year of the CSESAW (pronounced see-saw) workshop. The workshop was created to explore system level design synthesis and assessment capabilities for large, complex systems. These capabilities will facilitate the development of such systems from informal system requirements, through the design phase prototyping, and into implementation and post deployment. Component products produced by these capabilities are specifications that subenvironments, e.g., Hardware Engineering Environment (HWEE), Software Engineering Environment (SEE) and Human Computer Interaction Engineering Environment (HCIEE), will receive. The focus of the workshop is the development and integration of these varied technologies and the exploration of the creation of a system level engineering discipline, with support technologies that provide potential high payoff solutions to the difficult problems encountered by designers, developers, and maintainers of real-time systems. The emphasis is on resolving system level technology issues that cut across component boundaries, such as those associated with system behavior requirements of real-time, fault tolerance, cost, and security.

The focus of this year's workshop is synthesis for evolutionary systems. As technology has developed, computer-intensive systems have increasingly become extremely large and complex, controlling a wide variety of resources, and operating in many unforeseeable situations. Many of today's systems have hard real-time, stringent dependability, intensive security, and costly ownership requirements. They are typically implemented on a combination of parallel and distributed architectures and are embedded within a human organization structure and/or have human operators in the loop.

We welcome you to this year's workshop where we hope to provide an atmosphere in which the participants, including technology developers, researchers, users and customers, can meet, interact and exchange ideas on relevant issues. In the near future we hope to be able to say that this workshop was the beginning of a new focus on systems design and evaluation technologies.

This workshop would not have been possible without the hard work of many people, including the workshop, program, and advisory committees, authors, presenters of the submitted papers, panel members, workshop attendants, panel chairs, and breakout session chairs. A very warm "Thank You" is extended to all. In particular, we wish to acknowledge Michael Edwards, Ngocdung Hoang, Cuong Nguyen, Michael Jenkins, Kathy Lederer, Adrien Meskin, and Bill Farr. A particular thanks goes to Harry Crisp, the Program Manager, and to Elizabeth E. Wald of the Office of Naval Research (ONR) for their support and dedication in enabling the technology developments in this important area. Also, a thanks goes to CDR Gracie Thompson, formerly of ONR, for her help in providing perspective for researchers in the technology area. Finally, we would like to give a special thanks to Phillip Q. Hwang, who had the vision to initiate this series of workshops.

We are hopeful that our workshop will be merged into the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS) as a conference track. This will further the exposure of this important research area to the greater community.

Have a productive and enjoyable workshop!

Steven L. Howell
Workshop General Chairman

W. "Mac" McCoy
Workshop Assistant Chairman

## *ABSTRACT*

CSESAW '94 is exploring system level design synthesis and assessment capabilities for large, complex systems. These capabilities will facilitate the development of such systems from informal system requirements, through the design phase prototyping, and into implementation and post deployment. Component products produced by these capabilities are specifications that subenvironments will receive. The focus of the workshop is the development and integration of these multiple technologies and the exploration of the creation of a system level engineering discipline with support technologies to provide potential high payoff solutions to the difficult problems encountered by designers, developers, and maintainers of real-time systems. Further emphasis is on resolving system level technology issues that cut across component boundaries, such as those associated with system behavior requirements of real-time, fault tolerance, cost, and security. Specifically, the theme of this year's workshop is system engineering synthesis for evolutionary systems. Issues addressed within the workshop include requirements specification, requirements traceability, design capture, design evaluation, design optimization, security engineering, and dependability engineering. Panel discussions will emphasize essential domain models for long-life evolutionary systems, new paradigms in system engineering, and early evaluation metrics of system developments.

## *AGENDA*

1994 Complex Systems Engineering Synthesis and Assessment Technology Workshop
(CSESAW '94)
July 19-20, 1994

Holiday Inn
4095 Powder Mill Road
Beltsville, Maryland 20705

**Tuesday, 19 July 1994**

0730 Registration

0830 *Welcome*
Steven Howell, Naval Surface Warfare Center Dahlgren Division

**0850 SESSION I**

*Towards a Pre Requirements Specification Traceability Model*
Bala Ramesh, Naval Postgraduate School

*An Integrated Semantic and Syntactic Framework for Requirements Traceability Experience with* System *Level Requirements for a Large Complex Multi-segment Project*
James D. Palmer, George Mason University

*Implementation of NSWC Requirements Traceability Models*
John Nallon, TD Technologies, Inc.

*DOORS to the Digitized Battlefield: Managing Requirements Discovery and Traceability*
Nancy Rundlet, Zycad Corporation; and William D. Miller, AT&T Bell Laboratories

*Formal Requirements Specification Languages for Real-Time Systems: Expressibility Issues*
Ralph D. Jeffords, Naval Research Laboratory

1030 COFFEE

**1040 SESSION II**

*RECAP: A REquirements CAPture Tool for Large Complex Systems*
Michael Edwards, Naval Surface Warfare Center Dahlgren Division

*Tracing Product and Process Information when Developing Complex Systems*
Stephanie White, Northrop Grumman Corporation

*System Dependability Assessment Tool*
Eric W. Brehm, Advanced System Technologies, Inc.

*Synthesis of Multilevel Security and Timing Requirements in Complex Real-Time Database Systems*
Sang H. Son, University of Virginia

1200 Lunch

**1300 SESSION III**

*Tradeoff Areas for Secure System Development*
Catherine Meadows, Naval Research Laboratory

*Myths about Dependability in Complex Systems*
Michelle McElvany Hugue, Trident Systems, Inc.

*A Step Toward Integrating Dependability Concerns into the Systems Engineering Process*
Richard C. Scalzo, Naval Surface Warfare Center Dahlgren Division

*Testing and Fault-Injection of Distributed Protocols*
Farnam Jahanian, University of Michigan

1420 COFFEE

**1430 SESSION IV**

*The Synthesis of Software Artifacts with Implementation: Re-creating Requirements Specifications*
Evan Lock, Computer Command and Control Company

*Software Tools for Formal Specification and Verification of Distributed Real-Time Systems: A Progress Report*
J. Choi, University of Pennsylvania

*An Environment to Support Design Structuring*
Jee-In Kim, Computer Command and Control Company

*An Approach to Formalize Cooperative Intelligent Information Systems*
Naoufel Kraiem, University of Paris I

1600 COFFEE

**1615** **ESSENTIAL DOMAIN MODELS PANEL**
Chair: Evan Lock

**Wednesday, 20 July 1994**

**0800** **SESSION V**

*Distributed Synthesis Tools for Mission-Critical Computer Systems*
Parameswaran Ramanathan, University of Wisconsin-Madison

*Complex System Optimization*
J. Pukite, DAINA

*DRTSS: A Simulation Framework for Complex Real-Time Systems*
Jane W.-S. Liu, University of Illinois

*Application of the Analytic Hierarchy Process to Complex System Design Evaluation*
Michael L. Talbert, Virginia Polytechnic Institute and State University

*Modeling Computer Architecture for Surface Ship Combat Systems*
Andrew Mittura, SYSCON Corporation

*Using Metrics to Control and Predict the Quality of Space Shuttle Flight Software*
Norman F. Schneidewind, Naval Postgraduate School

1000 COFFEE

**1030** **EARLY EVALUATION PANEL**
Chair: Jane Liu

1200 LUNCH

**1300** **SESSION VI**

*Model-Based Synthesis of Complex Embedded Systems*
B. Abbott, Vanderbilt University

*Economics of Resource Allocation*
Carlos C. Amaro, New Jersey Institute of Technology

*Application of Uncertainty Management System (UMS) Techniques to the Quantification of Confidence, or Belief, in Complex Systems*
Tim Ramsey, Nichols Research Corporation

*Integrating Cost Models with Systems Engineering Tools*
Thomas C. Choinski, Naval Undersea Warfare Center

1420 COFFEE

**1430 SESSION VII**

*Software Assembly for Real-Time Applications* Based *on a Distributed Shared Memory Model*
David B. Stewart, Carnegie Mellon University

*Synthesis of Future Submarine Command and Control System Architectures*
Steve Harrison, Naval Undersea Warfare Center

*A Visual Platform for the Synthesis of Complex Systems*
Michael Gorlick, The Aerospace Corporation

*A Process-centered Methodology for Engineering of Complex Systems*
Azad M. Madni, Intelligent Systems Technology, Inc.

*A Methodology Framework for Optimal Design of Real-Time Dependable Computer Systems*
K. H. (Kane) Kim, University of California, Irvine

1615 COFFEE

**1630 NEW PARADIGM PANEL**
Chair: Stephanie White

*CONTENTS*

# TOWARDS A PRE REQUIREMENTS SPECIFICATION TRACEABILITY MODEL

Bala Ramesh
Naval Postgraduate School

## I. INTRODUCTION

A primary concern in the development of complex, large-scale, real-time, computer-intensive systems is ensuring that the design of the system meets the user needs. In the current environment of complex and constant changes in the fundamental role of the DoD, mission and operational needs evolve, necessitating reevaluation of the requirements of systems that satisfy them. Due to the size and complexity of these systems, the entire systems development process has become quite challenging to manage. Funding, time, and personnel are often at a premium, as well as technological resources. Advancements in computer technology encourage increased tasking of systems designers. In such a context, it is essential to maintain the traceability of mission and operational needs to requirements, ensuring that the system meets the current set of organizational needs.

Requirements traceability refers to "the ability to follow the life of a requirement, in both forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases" (Gotel and Finkelstein, 1993). Current practices in requirements traceability in the DoD address traceability essentially only after the requirements specification phase of systems development. The DoD currently spends approximately four percent of the total life cycle costs on requirements traceability efforts in large scale systems development. As current DoD standards that require traceability do not clearly specify what information should be captured and used, the practices and usefulness of traceability vary considerably across systems development efforts (Ramesh and Edwards, 1993).

Recent research suggests that a primary reason for problems with current traceability efforts is the lack of pre-requirements specification traceability (Gotel and Finkelstein, 1993). Current approaches treat requirements specifications as a black box and do not address the issue: **Where do requirements come from?** In the DoD environment, the lack of clear answers to this very same question severely jeopardizes development of new systems and consolidation of existing systems. For instance, the inability trace operational environments to operational capabilities and back necessitates ad-hoc fixes to systems in joint warfighting environment. During the Desert Shield and into the Desert Storm, Joint Warfare Commanders selected a series of warfighting vehicles, communications, and personnel from throughout the services to engage an enemy. Nearly six months of preparation was required to adequately put into place the mechanisms of war. In the absence of traceability, ad-hoc changes to various systems (such as the CAFMS, JSTARS) were required for various operational environments.

Pre-Requirements Traceability will assist the Joint Warfare Commander in matching their needs to existing systems and those under development. DoD has a well defined requirements generation process that produces information needed to provide such a traceability. However, the linkages or relationships among various phases of requirements generation, from the mission

area analysis to the development of operational requirements and system specifications are not explicitly captured. This severely restricts the Pre-RS traceability.

## II. PROBLEMS WITH CURRENT APPROACHES

The need to provide traceability is recognized in standards that regulate the development of systems for the U.S. Government; yet, there is no clear definition of the types of information or relationships between the various system components that are part of a traceability model. Neither the standards that require traceability as a part of any systems development effort nor the current literature elaborate on the specific types of traceability linkages to be maintained. Though current tools provide mechanisms to represent various types of linkages between system components, the interpretation of the meanings of such linkages is left to the user. Finally, the focus of the majority of the literature reviewed catered to traceability at the level of software design, rather than at the level of system design.

To effectively prove requirements compliance, requirements traceability should also address how the requirements are arrived at as well as the rationale that identifies not only the decisions, but also the supporting/opposing reasons behind those decisions (Ramesh and Edwards, 1993, p. 257). In systems development, the objective of requirements traceability is to ensure that the system meets the expectations of the user. It is especially important in the context of evolving user needs necessitated by strategic changes.

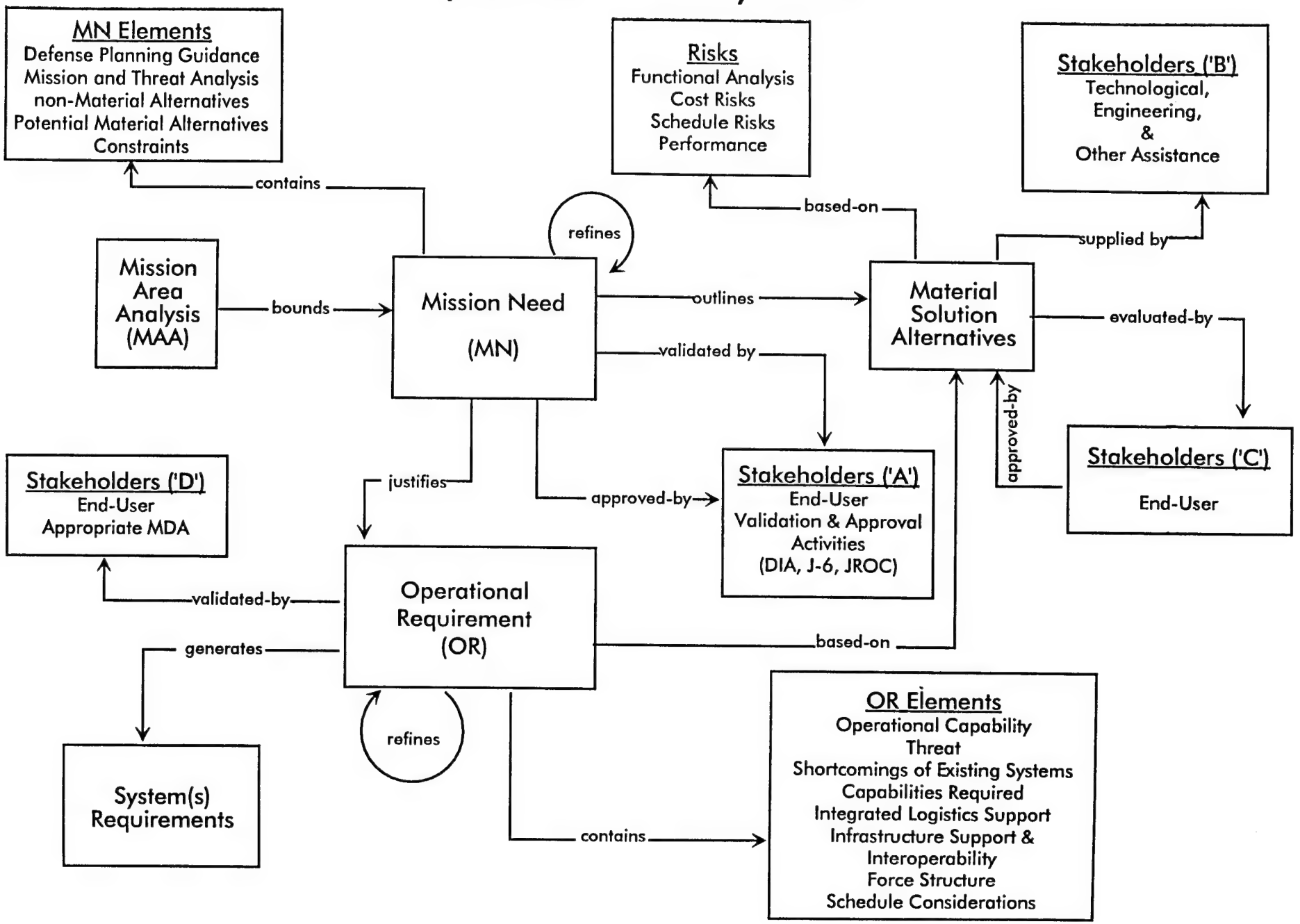
## III. TRACEABILITY IN DoD

In a Joint warfighting environment, the various Services must perform as a single cohesive entity. The Joint Warfare Commander should have the capability to trace their warfighting needs to the various Services systems capabilities, extracting out their systems needs for a given operational environment.

The Services have systems that the Joint Commander will require. Each of these systems must interoperate with other systems, often developed by other services and dissimilar in nature. Traceability can provide a means of analyzing the interactions amongst various dissimilar systems that comprise a Joint Commanders warfighting mechanism, enabling the him/her to make sound decisions on which systems will or will not operate in a Joint Environment.

Historically, the Services have procured Service unique systems. The current Requirements Generation Process outlined in DoD 5000 series (Defense Acquisition), requires the Services to analyze their systems procurement for Joint Service potential. Defense Information Systems Agency, Joint Interoperability Engineering Organization (DISA/JIEO) is the agency responsible for Joint requirements testing and evaluation. JIEO is concerned with interoperability and redundancy issues within the Services and Agencies that comprise the DoD systems development arena. Traceability could assist JIEO teams in organizing the Services unique requirements and analyzing similarities. For instance, a major issue that the JIEO is facing currently is the integration of Functional Process models of "similar" systems in use by

# pre-RS Traceability Model



various services so that joint systems could be developed. In the absence of traceability of the mission/operational needs and the rationale behind these models, the JIEO is finding the task impossible. Another example of similar problems was observed in the Computer Assisted Force Management System (CAFMS) during the Gulf war. The system was developed by the Air Force to coordinate aircraft tasking issues. During the war, all the services were trying to use information from CAFMS, but only the Army, Air Force and Marine Corps were prepared to utilize the system. Navy could not effectively use CAFMS afloat, and is still generating requirements for their version of CAFMS. Lower echelons of the Army were not able to obtain vital CAFMS information in a timely manner. Traceability among the system requirements and the joint need, if captured during development of this system, would have made connectivity between the services possible and the ad-hoc patchwork that was used to share information could have been avoided.

The objective of our research is to develop a model of pre-requirements traceability. The model is based on empirical studies of the needs of various stakeholders involved in the requirements generation during the initial development of large scale joint systems. The data collection techniques used in this research include focus group discussions and case studies of joint systems development activities. Our work focuses on the concept development stage of systems development, from the Mission Area Analysis to the successful Milestone 1 approval in the DoD systems acquisition life cycle.

## IV. A MODEL FOR PRE-RS TRACEABILITY

A major challenge in this research is the development of a model that depicts the semantics of multiple traceability linkages between system components during initial systems development. Components can be described as tasks, agents, inputs, and outputs of the development process. Linkages describe the relationships between components. The model was developed based on an analysis of focus group discussions and interviews.

In this section, traceability linkages will be distinguished by uppercase, bold faced letters (LINKAGES), while components that they link are shown with uppercase, italic letters(COMPONENTS). For every link in the model an inverse may be defined. The following is a discussion of the pre-RS traceability model presented in Figure 1.

### 1. Mission Need

*MISSION AREA ANALYSIS* is the determination and exploration of the environmental and strategic needs based on future technologies, threats, and organizational goals. **BOUNDS** on what the *MISSION NEED* should and should not include are established by *MISSION AREA ANALYSIS*. The environment of the cold war and the possibility of nuclear war bounded most systems to the Soviet threat from the 1940s until recently. *MISSION NEED* is the operational capability to meet a deficiency found with regard to the strategic and environmental needs ("...the mission needs statement is based on the operational requirements documents so its a process.")[1] For example a *MISSION NEED* could suggest development of an automated system

to generate a comprehensive report including target, take-off, landing, and fuel information to fulfill the strategic and organizational needs of a military service.

*MISSION NEED* is **VALIDATED-BY** *STAKEHOLDERS ('A')*. The link **VALIDATED-BY** refers to the determination of whether the *MISSION NEED* meets the strategic and environmental needs as defined and understood by the *STAKEHOLDERS*. For instance, the draft Mission Needs Statements (MNS) is sent to the Joint Interoperability and Engineering Organization (JIEO) of the Defense Information Systems Agency (DISA). JIEO validates the draft MNS with regards to MNS ability to meet its joint needs as defined by the Joint Requirements Oversight Council (JROC) ("....we receive draft and approved MNS and ORDs (Operational Requirements Documents) from the services and subsequently staff them out throughout JIEO centers, CINCs, Services, and Agencies and provide an assessment on interoperability, capability, and integration.") *STAKEHOLDERS* ('A') are those organizations that have validation responsibility for, or have a vested interest in, the *MISSION NEED*.

*MISSION NEED* is **APPROVED-BY** *STAKEHOLDERS ('A')* so that the *MISSION NEED* expresses the operational need of the*STAKEHOLDERS ('A')*. ("For Joint Services systems, the recommendation for approval is given to the Joint Requirements Oversight Council and this recommendation is an important wicket which the services must pass.") This illustrates that approval of a MNS by JIEO is required for further development of a system when interoperablity is a strategic need. *STAKEHOLDERS ('A')* are those organizations that have approval responsibility. DISA JIEO is a *STAKEHOLDERS ('A')* because it has to approve the MNS and acts to insure the interests of the Joint Chiefs of Staff (JCS) via the JROC.

*MISSION NEED* **CONTAINS** *MISSION NEED ELEMENTS*. Analysis and planning to meet the strategic and environmental needs are contained within the *MISSION NEED*. Examples of *MISSION NEED ELEMENTS* are Defense Planning Guidance, Mission Analysis and Threat Analysis. *MISSION NEED ELEMENTS* also include material alternatives(i.e. systems) and nonmaterial alternatives (i.e. changes in procedures or policy). *MISSION NEEDS* expressing a material alternative discuss the nonmaterial alternatives explored. A specific example is that the Marine Corps Combat Development Command (MCCDC) has developed Doctrine which is incorporated into the MNS for systems that involve the Marine Corps. *MISSION NEED* **REFINES** *MISSION NEED*. The MNS for DoD systems is reworked to meet the organizational needs and concerns for the Joint Chiefs of Staff.

**2. Material Solution Alternatives**

*MISSION NEED* **OUTLINES** *MATERIAL SOLUTION ALTERNATIVES* which are material options that are capable of meeting the operational need as defined by the *MISSION NEED*. An example of this are the alternatives explored by DoD for the FA-18 E/F program. The program matured from a study called Hornet 2000 which explored *MATERIAL SOLUTION ALTERNATIVES* for a export model of FA-18 aircraft in 1987. ("There were three major configurations developed for this project and there were subconfigurations. Anyway configuration 3C looks a lot like what the aircraft looks today.") The *MISSION NEED* **OUTLINES** or provides the guidelines for a search of possible solutions to meet operational

need. The DoD MNS is the basis for developing trade studies on the operational need. ("There are a lot of trade studies done in a cost and evaluation kinda phase before the actual ORD.")

*MATERIAL SOLUTION ALTERNATIVES* are **BASED-ON** *RISKS.* The *RISKS* associated with solutions provide the basis for *MATERIAL SOLUTION ALTERNATIVES. RISKS* are the dangers and hazards associated with technology, performance, and costs. For example the Navy program for an advanced attack aircraft A12 was canceled because the program costs became excessive due to the *RISKS* associated with it. The evaluations of the *RISKS* associated with cost, threat, and performance are considered essential.

*MATERIAL SOLUTION ALTERNATIVES* are **SUPPLIED-BY** *STAKEHOLDERS ('B').* E.g. The DoD research and delvelopment laboratories provide *MATERIAL SOLUTION ALTERNATIVES* to the Definition and Documentation Activity. ("I dangled the Alternatives in front of them along with the rough costs and what kind of capability they could get for how much money and when.") *STAKEHOLDER('B')* are those organizations providing possible *MATERIAL SOLUTION ALTERNATIVES* to meet the operational need. e.g. Private and public research laboratories. *MATERIAL SOLUTION ALTERNATIVES* are **EVALUATED-BY** *STAKEHOLDERS ('C')* as to how to determine the *MATERIAL SOLUTION ALTERNATIVES* meet the operational need. *STAKEHOLDERS ('C')* are those organizations and entities that assess the *MATERIAL SOLUTION ALTERNATIVES* . An example is that CNO staff members evaluates the different options and alternatives provided.

*MATERIAL SOLUTION ALTERNATIVES* are **APPROVED-BY** *STAKEHOLDERS ('C')* to ensure that they meet the operational need as defined and understood by the *STAKEHOLDERS('C')* ("...there are several alternatives to go about doing it. We recommend, in order of priority, this way; this way; this way; based on risk. CNO then comes back and says "Okay, from your DOP [Chap 3.B.5] we going to take option X.") The *STAKEHOLDERS('C')* are those organizations and entities that approve the *MATERIAL SOLUTION ALTERNATIVES.*

**3. Operational requirement**

*OPERATION REQUIREMENT* is **BASED-ON** *MATERIAL SOLUTION ALTERNATIVES. OPERATIONAL REQUIREMENT* is the requirement developed to meet the operational need as developed by the *MISSION NEED.* For DoD the Operational Requirement Document expresses the *OPERATIONAL REQUIREMENT* for a system **BASED-ON** (i.e., developed from or supported by) the *MATERIAL SOLUTION ALTERNATIVES.* ("So, the ORD for this program became a derived document based on these studies").

*OPERATIONAL REQUIREMENT* **CONTAINS** *OPERATIONAL REQUIREMENT ELEMENTS.* For DoD systems *OPERATIONAL REQUIREMENTS* must embody the force structure, logistic considerations, threat, and operational capability. *OPERATIONAL REQUIREMENT ELEMENTS* are constraints that mold the *OPERATIONAL REQUIREMENT* and define standards which the *OPERATIONAL REQUIREMENT* must adhere to. DoD standards provide the structure and the ORD must include them ("MOP( Memorandum of Policy) 6212 says that...a standards profile is supposed to be part of the ORD.")

*OPERATIONAL REQUIREMENT* is **VALIDATED-BY** *STAKEHOLDERS ('D').* The **VALIDATED-BY** link refers to the determination of whether the *OPERATIONAL REQUIREMENT* adheres to the operational need as defined and understood by the *STAKEHOLDERS('D'). STAKEHOLDERS('D')* are those organizations providing oversight that the *OPERATIONAL REQUIREMENT* meets the operational need that was developed in the *MISSION NEED.*

*OPERATIONAL REQUIREMENT* is **APPROVED-BY** *STAKEHOLDERS ('D').* For DoD the ORD must beapproved according to law by the appropriate Milestone Decision Authorities. *STAKEHOLDERS('D')* are those organizations and entities that approve the *OPERATIONAL REQUIREMENT.* Further, *OPERATIONAL REQUIREMENT* **REFINES** *OPERATIONAL REQUIREMENT. MISSION NEED* **JUSTIFIES** *OPERATIONAL REQUIREMENT* so that the *OPERATIONAL REQUIREMENT* must maintain and assert the operational need the *MISSION NEED* expresses. The *OPERATIONAL REQUIREMENT* **GENERATES** or creates and brings into existence the SYSTEM REQUIREMENTS. SYSTEM REQUIREMENTS are system specific requirements that are developed to meet the *OPERATIONAL REQUIREMENT.*

## V. DISCUSSION

Our study of the DoD requirements management process suggests the need for and the feasibility of creating a pre-requirements specification traceablity scheme. Our model will be refined based on analysis of additional data.

## REFERENCES

Gotel, O and Finkelstein, A. " An Analysis of the Requirements Traceability Problem", Imperial college technical report (Unpublished), London, UK.

Ramesh, B., and Edwards, M., "Issues in the Development of a Requirements Traceability Model," in proceedings of IEEE International Symposium on Requirements Engineering, San Diego, CA, January 1993.

[1]This is a direct quote from a focus group.

**An Integrated Semantic and Syntactic Framework for Requirements Traceability**

**Experience with System Level Requirements**

**for a**

**Large Complex Multi-segment Project**

1994 Complex Systems Engineering Synthesis and Assessment Technology Workshop (CSESAW '94)
July 19-20, 1994
Washington D. C.

James D. Palmer and Richard P. Evans

James D. Palmer
BDM International Professor of Information Technology
Director, Center for Software Systems Engineering
ST II, Room 100
George Mason University
4400 University Drive
Fairfax, VA 22030-4444

(703) 993-1504 Office
(703) 993-1521 Fax
jpalmer@gmu.edu

Richard P. Evans
ST II Room 100
George Mason University
4400 University Drive
Fairfax, VA 22030-4444

(703) 993-3724
revans@gmu.edu

## Abstract

*The CSESAW '93 Panel on Traceability cited some of the challenges in achieving traceability, and noted that there were no automated techniques available to address those needs. One of the particular challenges to providing traceability to and from system level requirements is that it becomes necessary to utilize both the constructs of language semantics as well as syntax. Language semantics are needed to assure the trace is related to the meaning or context of the requirement or set of requirements, while syntax is necessary to trace to a specific word or phrase, without regard to meaning or context. Integration of both constructs is required to provide for full requirements traceability for natural language requirements statements. This paper describes an integrated semantic and syntactic assessment framework for requirements traceability and experiences in applying that framework for the traceability assessment of requirements for large complex multi-segment program.*


## Introduction

Typical of the currently available automated (or semi-automated) approaches to requirements traceability are those that provide for traceability through a variety of syntactic language components: hypertext linking, unique identifiers, or syntactical similarity coefficients or combinations of these. In the hypertext linking construct, the "hotword" or word/phrase to be linked to other requirements is manually identified and entered into the hypertext tool. Links are automatically made and maintained by the tool to provide forward and reverse traceability for the word selection. In the unique identifier approach, an identifier is assigned that remains with the individual requirement throughout the life of the project. To assure traceability, this unique identifier provides a "fan-out" capability within a hierarchical structure such that one system level ("A" level) requirement may be the parent to many "B" level requirements which, in turn, may be the parents for great numbers of "C" level requirements. Use of syntactic similarity coefficients ascertains whether or not a pre-defined number of words of a given requirement are found in another requirement. When the coefficient is above this pre-defined threshold, the two requirements in question are said to trace.

There are problems with each of these syntactic language component approaches. They do not consider the semantics or context in which the tracing is to occur. Hypertext linking finds the search text without regard to the placement in the text and without regard to the way in which the words are used. Use of a unique identifier provides access only to those requirements so identified with no perspective as to meaning or context. Syntactic similarity coefficient traceability is like hypertext linking in that it is indiscriminate as to the meaning and context of the requirement to be traced. There are many challenges to providing integrated semantic and syntactic capability for automated support to natural language requirements traceability. To meet these support needs, there are particular challenges that include: (1) processing of tables and graphics as these are particularly integral to traceability from informal text to more precise semi-formal/formal diagrammatic representations; (2) capture of the assumptions and rationale applied in traceability, and recording this feedback and updates to the original requirements; (3) traceability scalability issues for natural language requirements; and (4) providing a human computer interface for interactive capabilities.

In this paper we present a brief description of the Advanced Integrated Requirements Engineering Environment (AIRES), a system and software requirements support environment that handles natural language text from requirements documents. AIRES provides the necessary processes and methods to support the use of semantic and syntactic constructs for requirements traceability. Next, a review of the AIRES integrated semantic and syntactic assessment framework for traceability is discussed. Following that, this approach is applied to a large complex multi-segment program and the specific semantic and syntactic tracing procedures are described. Finally, experiences and outcomes derived from the use of the assessment framework are supplied.

## AIRES

AIRES provides three frameworks (Process, Methodology, and Database) to support the activities necessary for the development, assessment, and archival of system and software requirements in natural language. These frameworks are integrated yet enable independent application of current and in-development methodologies

(separately and in combination for all forms) for automated support. This automated support is provided to the Process Framework, process, elicitation, assessment, and transformation, through the Methodology Framework, an assessment framework that manages the various methods. All information is archived in the Database Framework, the integrated database, which provides access to files such as original requirements text, methods used for assessment, procedures used to apply combinations of methods, or various products that result from AIRES applications. A conceptualization of the AIRES architecture and the constituent frameworks is depicted in Figure 1.

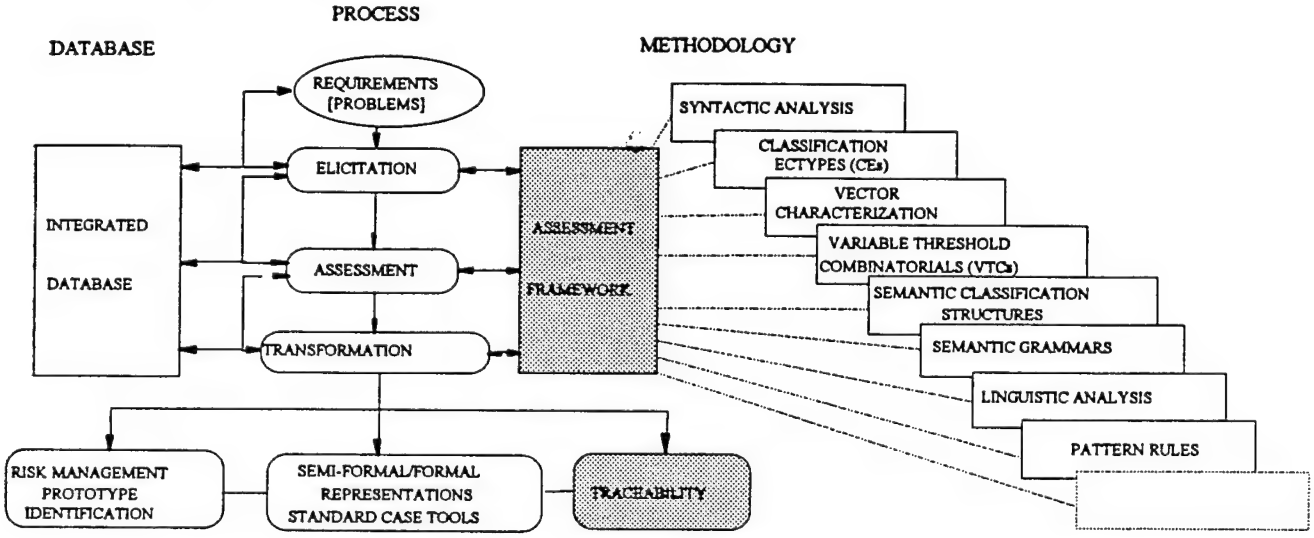



Figure 1. AIRES Architecture: Process and Methodology with Integrated Database

To support requirements traceability, it is necessary to span each of the AIRES features: (1) integration of the three frameworks to assure access to legacy systems, newly captured concepts, and the methods to perform assessments; (2) use of sets of semantic and syntactic methods and procedures, as necessary, to assure complete traceability; and (3) a common database for archival and retrieval of pertinent information. As noted earlier, requirements traceability necessitates the integration of semantic and syntactic language constructs and AIRES methods incorporate a variety of such semantic and syntactic assessment tools to provide automated support to requirements traceability. Methods include innovative features such as semantic grammars, vector calculation of syntactic similarity coefficients, syntactic parsing, and linguistic analysis. As may be seen from Figure 1, the capabilities of the Assessment Framework are utilized for each of the three AIRES Process activities, elicitation, assessment, and transformation. The Assessment Framework provides a single home for each of the methods of the AIRES methodology. The key to providing automated support to requirements traceability for natural language requirements statements is the application of methods managed by the Assessment Framework. To provide an in-depth insight to requirements traceability the AIRES integrated semantic and syntactic assessment framework is next discussed.

### Integrated Semantic and Syntactic Assessment Framework

The AIRES integrated semantic and syntactic assessment framework, as shown in Figure 2, consists of four constructs: (1) CEs; (2) master CEs comprised of CEs that may be of different forms and/or the same form with variations in one or more of their rules and tables; (3) CE Procedures (CEPs) that control the application of master CEs; and (4) meta-CEPs that control the application of CEPs.

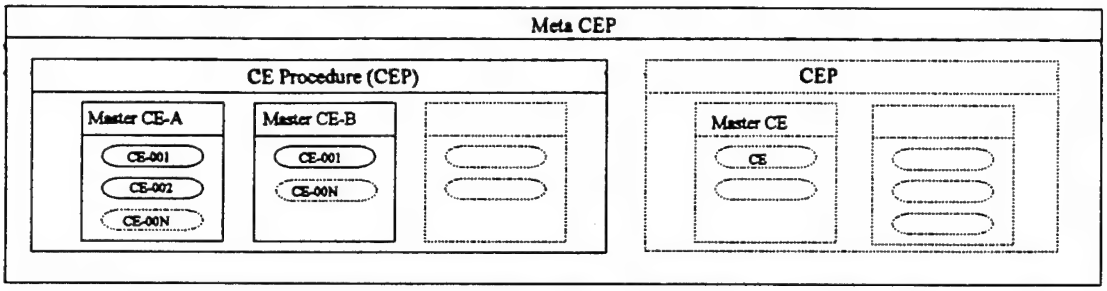



Figure 2. Assessment Framework: CEs, Master CEs, CEPs, and Meta CEPs

A CE is a *classification ectype* (an ectype is a representation). They are text-edited files that include a combination of *rules* and *tables*. As the basic element of the assessment framework, CEs enable the categorization of requirements where categories are defined as elements of a classification structure, as illustrated in Figure 3.

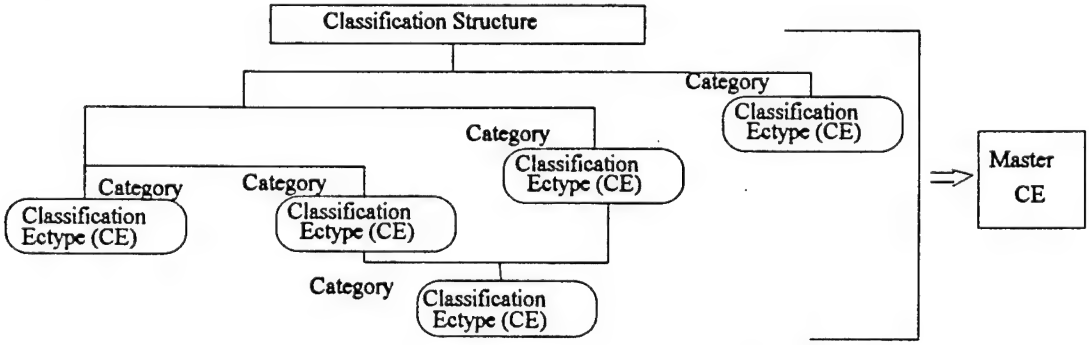



Figure 3. Classification Structure, Categories, and Classification Ectypes (CEs)

There are two forms of CEs, they are distinguished by the form of their *tables*: (1) *word form* CEs, and (2) *integer form* CEs. Word form CEs define term feature parameters such as parts of speech (separately and in combinations such as co-occurring subject noun and object noun phrases), case sensitivity, degree of term juxtaposition (phrase for immediate proximity, word separation limits, or within same sentence or IDs), order constraint, weighted synonyms, and other tables of designated data such as records of changes which may be used as measures of volatility or designated traces to other IDs in the same or separate documents. Integer form tables use either requirement identification numbers, called IDs, or pair-wise requirement similarity coefficients called SCs. The first are called *ID-Based* integer form CEs, the second are called *SC-Based* integer form CEs. Each of the CE forms have separate rule forms. Rules for ID-based integer form CEs apply the logical operations of *and, or, not,* or *exclusive or* or combinations of these to categories established by either of the other CE forms. SC-based integer form CEs enable categorizations by the definition of parent-children sets of requirements. The calculation of similarity coefficients (SCs) is as a syntactic pair-wise dot product measuring common terms in each of two IDs. That calculation includes the use of weighted words, weighted synonyms, and a set of optional stemming operations. CEs, in these separate forms, are the basic building blocks of the assessment framework of AIRES.

### Large Complex Multi-Segment Project: Experience and Results

The assessment task for a large complex multi-segment project was to ascertain specific requirements that required a trace and failed. The system requirements were established in three segment requirements documents and an associated Interface Control Document (ICD). Each segment document included an approximately equally number of requirements, with a combined total in excess of 1,000 paragraphs. Each requirement ID had from 3-20 sentences. Both internal non-traces as well as non-traces across documents were to be identified.

The meta-CEP used to identify non-traces included a *cross-referencing-based* CEP and a *title-based* CEP, as shown in Figure 4. Each of these two CEPs applied both syntactic and semantic master CEs in specific combinations. The requirements identified as non-tracing in each CEP were compared with each other in a truth table approach: any known trace identified by one CEP negated the categorization as an incomplete trace in the other.

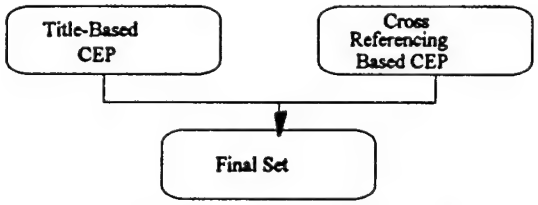



Figure 4. Meta-CEP for Identifying Non-Traces

The steps of the title-based CE Procedure (CEP), as one part of the meta-CEP for traceability, are shown in Figure 5. The terms *base document* and *target document* designate documents containing all the requirements that should trace from a base to a target document. The required tracing assessment was both internal, the case for which the base and target documents are the same, and across the other three documents. All combinations of the four documents were considered.

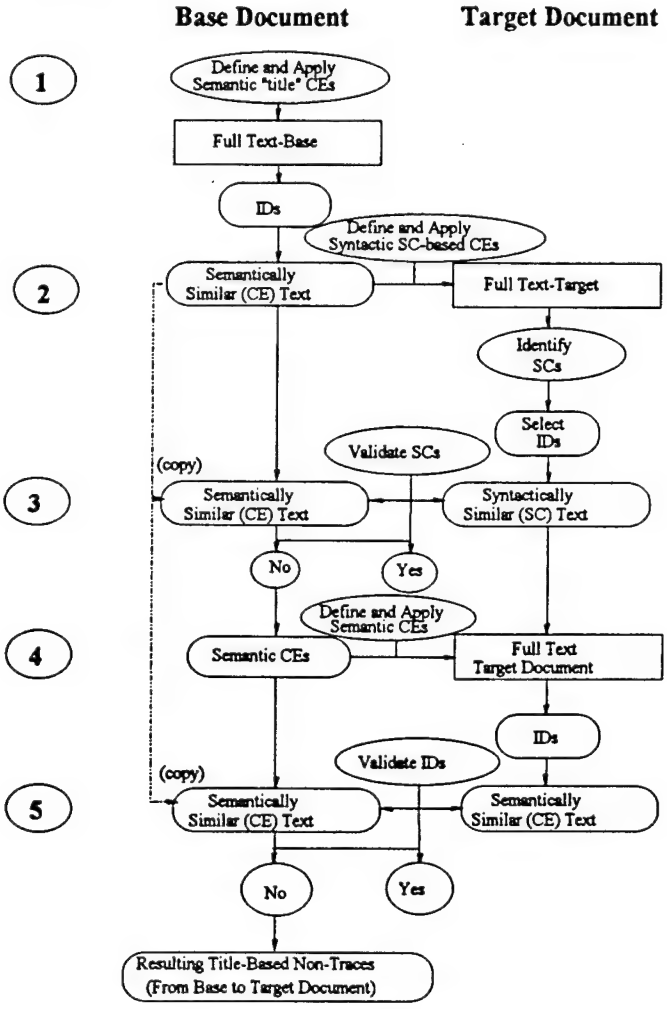



Figure 5. Semantic and Syntactic CEP for Non-Traces: Title-Based

The five steps of this title-based CEP include: (1) preparation of semantic word form CEs to produce master CEs and their application to the text of a base document; (2) application of the subset text, identified semantically by step (1), in a syntactic SC-based master CE against the text of each of the other three "target" documents (only one is shown here for simplicity of the diagram); (3) validation of the text identified syntactically by step (2); (4) preparation and application of more refined semantic CEs to form master CEs based on the subset text identified as not having a trace in the step (3) validation; and finally, (5), validation of the non-trace requirements identified in step (4).

The net result of the application of the full meta-CEP was the identification of 30 non-traceable requirements from the entire set of system requirements, both within and across the system segment documents and the ICD. The involvement of each document in either a non-trace internally or a non-trace involving one or more of the others is presented in Figure 6.

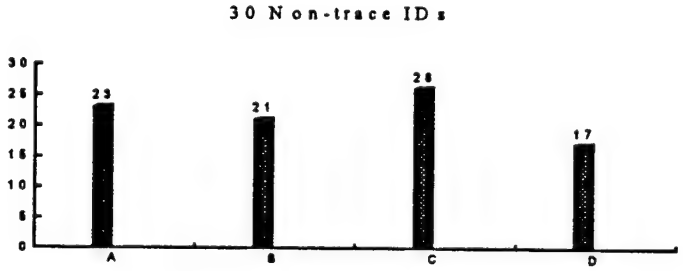



Figure 6. Involvement of Documents A, B, C, and D with Non-Tracing Requirements

This assessment experience in addressing non-traces both internal to a given document as well as across sets of documents in all combinations demonstrated (1) the essentiallity of automated support to the traceability of natural language requirements in large complex multi-segment systems, and (2) the efficacy of the AIRES integrated semantic and syntactic framework for that needed automated assistance. A byproduct of this assessment, that was directed to the identification of non-traces, was the identification of many requirements that included traces with multiple origins/destinations. The challenge of assessing that positive tracing, on multiple dimensions, demonstrated the added need for an integrated semantic and syntactic framework for automated assessment support.

## Summary

The AIRES assessment framework and techniques for the integrated application of both semantic and syntactic rules provide for effective, efficient, and comprehensive identification of non-traceable requirements in large complex multiple segment systems. The framework provides for the categorization of requirements in classification structures by a diverse combination of CEPs, each of which applies unique combinations of both semantic and syntactic CEs. CEs, as rules and tables for the categorization of requirements, serve as the basic building blocks of the assessment framework and may be applied in CEPs either singly or in combination as elements of master CEs. As text-edited files, CEs, master CEs, CEPs, and meta CEPs are constructed without the creation of new code. That isolation of framework operational variables from software eliminates inevitable coding delays, restrictions in operational flexibility, the potential corruption of stable operational code, and exacerbation of operational problem diagnostics by the introduction of code uncertainty as an additional potential problem cause.

The traceability experience and results with the requirements for a large complex multi-segment project, identifying 30 non-tracing requirements out of a set of over 1,000 requirements paragraphs distributed within and across four separate requirements documents, demonstrated (1) the need for automated support to traceability, and (2) the effectiveness of the AIRES framework for the integrated application of semantic and syntactic assessment techniques for traceability. That experience also identified the need for effective automated support to traceability on multiple origins/destinations dimensions.

# Implementation of NSWC Requirements Traceability Models

John Nallon
TD Technologies, Inc.
Suite 200
2425 North Central Expressway
Richardson, Texas 75080

In Collaboration With
Mr. Michael Edwards
Naval Surface Warfare Center

## Abstract

Understanding the relationship between system requirements and the components that make up the design and implementation of a system is a very important step in the system development process. As systems become larger and/or more complex, it becomes more apparent that comprehensive requirement traceability methods must be utilized to help understand the system requirements and their relationships to the system design elements. As these techniques are implemented, new problems arise. Displaying comprehensive traceability relationships becomes a real challenge in a graphical user interface. This became painfully evident when Texas Instruments, through their next generation systems engineering tool based on object oriented technology, began to implement the traceability models developed through the Office of Naval Research, Engineering of Complex Systems Technology Program. This paper discusses the issues and challenges that arose through their collaborative efforts, some of the initial work-arounds in solving these problems, the current status and future plans for SLATE, a System Level Automation Tool for Engineers.

# Introduction

As part of the Office of Naval Research, Engineering of Complex Systems technology (ECS) program, methods for specifying requirements and methods for complex traceability are being developed. The major thrust of the ECS program is to develop a better understanding of requirements. The ONR research has lead to a more structured approach to requirements traceability and new, but complex techniques for linking requirements from their source through all the threads in a system design from the top-most need to the lowest level of implementation.

## Specifying Requirements

In today's practice, requirements are typically specified in English prose documents which are often ambiguous and inconsistent. In order to alleviate some of the shortfalls of today's requirement management techniques, a more structured approach to specifying requirements is needed. Within a structure, each requirement can be captured individually along with appropriate attributes which classify and describe the requirement. This type of structure can allow for a clearer and more precise understanding of the requirements and pave the way for automated techniques that check for completeness and consistency within the requirements.

## Traceability Techniques

The current commercial state-of-the-art requirements traceability techniques use simple traceability links to relate a requirement to other requirements (usually from document to document) and to elements of the design that implement the requirement. The current methods and tools fall short on capturing the specific nature of relationships that exist between the requirement objects and their complying elements. In order to fully understand the complex relationships that exist within a complex system, more robust techniques for capturing traceability relationships are needed.

To obtain a better understanding of what relationships are needed for doing complex traceability, the Naval Postgraduate School (NPS) conducted a number of studies to develop complex requirement traceability models. Initially, the study consisted of graduate students at NPS participating in focus groups [Ramesh 92]. This study produced the first NSWC traceability model. Further focus groups were then conducted using practicing engineers from both DoD and Industry [Ramesh 93]. The analysis of these focus groups led to more complete requirements traceability models which show the complex relationships between the entities of the system design.

# TI System Overview

The System Level Automation Tool for Engineers (SLATE) is a multi-user, client/server, object-oriented application which provides a dynamic representation of an evolving system design using multiple information perspectives of objects stored in a design database. SLATE provides a variety of functions including requirements management, documentation generation and management, performance management, and report generation. Texas Instruments has incorporated the ECS program requirement traceability models and techniques into the SLATE database and the SLATE Requirements Management Subsystem to provide users a state-of-the-art information management capability for complex system analysis and designs.

## Requirements Management

The initial release of SLATE was focused on the initial tasks that systems engineers perform during the conceptual phase of a product, mainly Requirements Management and Documentation. Subsequent releases will address other needs in analysis, interfacing, test, simulation, and integrated downstream feeds to detailed design systems. The SLATE Requirements Management Subsystem design was based on numerous IEEE and NCOSE papers on the subject of Requirements Management [Requirements Management Using SLATE, 1994] as well as Mil-Std-499 A/B. The most influential work being the papers on traceability relationships and modeling that document the recent work of the Naval Post Graduate School and the Naval Surface Warfare Center [Ramesh, Edwards, 1993]. The NSWC requirement traceability models proved to be an independent verification of the approach taken for SLATE Requirement object class design, the Requirements Management System, the requirements traceability features, and the definition of the attributes on the object classes associated with documentation and traceability. In addition the models have added several dimensions to the SLATE design that will be provided in future releases.

## Requirements Management Approach

Systems engineering is challenged with numerous tasks while trying to manage requirements. Communication with the customer, establishing the customer needs, eliciting and indentifying requirements, analyzing the requirements, allocation and flowdown of requirements, establishing traceable audit trails, verifying the design meets the requirements and documenting everything in specifications for the design staff are the main tasks of systems engineers performing requirements management.

The current methodology for implementing requirements traceability tends to be from document to document and we get bogged down in the management of the documents and the traceability. This approach can be taken in SLATE if it is the preferred methodology of the users, however the SLATE Requirements Management System promotes a requirement to requirement traceability and treating documents as an output of the system. In this paradigm, the focus is shifted to managing the requirements and not the documents. Traceability is maintained in the design database independent of the document paragraphs or sections. Features are provided to store, create, modify and analyze the requirements in a "working pool of requirements" rather than inside a document. This approach improves productivity by reducing the amount of time being spent generating and managing documents.

## Identifying Requirements

Generally, system engineers will begin a program with an "A" level specification developed by a customer or proposal team. Assuming this document is provided in electronic form, SLATE imports ASCII or word processor native files into the SLATE object-oriented database as a Document object comprised of many Document Paragraph objects. In order to assist with identifying requirements from the document, a Requirements Identifier is used to break the document into Requirement objects. There are three levels of granularity available for parsing the document into Requirement objects. These are: Numbered Paragraph, Paragraph and Sentence level parsing. Additional capabilities are provided to set Requirement object attributes to known values and parse by a region of the document (section start and stop).

# SLATE User Interface

The SLATE User Interface is a multiple window, Motif-style graphical and textual interface featuring a direct manipulation Object/Action paradigm, context sensitive menus and multiple views of the design data stored in the database. The UIF is implemented with the Galaxy Application Toolkit from Visix Software, Incorporated. Windows are utilized to display objects and provide access to all attached design data and functions that can be performed on the objects. The SLATE Window types include the Project Window, Documentation Tree Window, Documentation Window, Report Window, and Diagram Window. The Requirements Management features of SLATE utilize the Project Window, Folder Window, Report Window and Diagram Window to display and organize the Requirement objects in the database.

## Object Classes

Design data is stored as instances of SLATE object classes in the object-oriented database. Hierarchical and non-hierarchical graphical or textual objects are utilized to develop and annotate a design in the SLATE database. Every object is tagged with session information as to when it was created, who has it locked for revision and who modified it last. There are numerous object classes in the SLATE database to support system engineering design activities. For the purposes of brevity we will contain our overview to the objects associated with requirements management.

## Requirement Objects

Requirement objects are non-hierarchical objects that capture requirement text, identification numbers, and extensive attribute values to assist with managing requirements. These objects are displayed in the Project Window or Folder Window.

## Requirement Traceability Links

Traceability links are provided to establish defining and complying relationships between objects and the flow of requirements in a pool of working requirements. Traceability links establish one to one relationship between objects. Transitional Maps (TRAMs) are special links that establish one to many, many to many, or many to one relationships

across multiple views of a design database. TRAMs (TRAnsitional Mappings) are relations which are one to many, one to one, or many to many. They allow the user to show how they went from one hierarchical design flow to another, such as going from the conceptual design to physical implementations. Links are also placed between defining objects and their complying object for traceability back to the original customer requirements.

### Abstraction Blocks

Abstraction Blocks are hierarchical objects that can represent any design element, an organization structure, Work Breakdown Structure, or any structure that can be represented by Parent-Child relationships. These objects are displayed in the Outline View, Tree View or Diagram View of all SLATE Windows except the Document Window.

### Notes

Notes are objects that can be attached to any SLATE object for the purpose of annotating the design. Note objects can be typed as Rationale, Assumptions, Decisions, Questions, Answers, Phone Call, Hall Talk, or any other user definable choice that identifies the information type or source. The Note Types described in the ECS requirement models are all included as choices in the SLATE database.

### Attributes

Attributes are attached to all SLATE objects and are used to provide a means of capturing specific information to characterize a design object. The Requirement objects have extensive attributes provided to capture data to support requirements management and report generation. The object attributes are single choice, multiple choice or fill-in text and are defined at the time the database is generated (a future release will have user-definable attributes).

### Ports, Paths, and Connections

Context, functional flow and data flow diagrams are created in the Diagram View. There are 4 types of objects which appear in the Diagram View: Abstraction Blocks, Ports, Paths, and Connections. Ports represent the direction of information. There are 3 types of ports - input output and bidirectional. Paths and Connections are the graphical line representing the flow in a diagram that share common ports.

## Use of ECS Research Information and Models

While researching requirements engineering information, the SLATE systems engineering team attended the 1993 IEEE Requirements Engineering Symposium. Of particular interest during that symposium was a paper presented by Michael Edwards (NSWC) on the empirical studies they had engaged with the Naval Postgraduate School {ramesh, edwards 1992]. After reviewing numerous papers on the subject of requirements from the 1993 symposium, we focused on the ECS research because it was in-line with the SLATE requirements traceability schema. The NSWC work saved us a lot of time developing the models and language of requirements so we could focus our resources on the design. The detailed information from the ECS research regarding relationship types and the language needed to support complex system requirements management has been incorporated or specified for future SLATE releases.

The ECS models were used as a reference to define or refine the Requirement, Attribute, Note object types, and relationship links between the SLATE objects. The most difficult implementation has been the relationship types and how to represent them in the user interface in an intuitive manner that met human factors guidelines and industry standards. The good, the bad and the ugly are terms often used to refer to our design history with object icons and relationship graphics. Some of these lessons learned are described and shared with you in the following paragraphs.

## The Difficulties

SLATE represents a paradigm shift in information management for engineers developing systems. As with any new technology or shift in paradigm there are many difficulties to overcome and in this case many were found to be culturally based. Many of todays practicing systems engineers obtained their knowledge without the use of computer aided engineering. They may, but probably do not have an engineering workstation. They typically have a personal computer

with a single user operating system and a discrete set of their favorite tools. They are an island of automation set in their patterns of solving problems. Changing methodology and culture are two of the most difficult problems SLATE has had to address. Developing icons to represent objects for a culture that isn't at ease with graphics interfaces has posed us many hurdles to overcome.

## Remaining Methodology Neutral

Every company and even multiple organizations within a large corporation have their own methodology. SLATE was specifically directed to be methodology neutral and not force a pre-defined methodology on the work place. SLATE has been designed to support an in place methodology or product development process, pre-defined procedures, approval cycles, configuration management processes and other methodology components. SLATE is also being used to assist with the development and improvement of product methodologies. The intent is for the methodology to be imposed procedurally in the use of SLATE and not for SLATE to impose a methodology and therefore possibly impede the user's workflow. This approach drove the software development to providing features that would allow more than one way for the user to manipulate the database to accomplish a task or represent a design. This neutral approach also created a 20 month argument over what icons should look like to represent data in the database.

## Maintaining Simplicity and Consistency in UIF

Practicing, experienced systems engineers were the driving force at Texas Instruments for a tool such as SLATE. It quickly became evident from the feedback from the users of the SLATE prototype that the User Interface had to be intuitive and not hostile or the users would not use the tool. The data must be consistent across all windows and the UIF must be as simple as possible. We found this to be a non-trivial problem for a multi-user, object oriented, client/server application designed to assist an engineer with the design of complex systems.

A Human Factors Engineer was assigned in the early design stages to address the usability, consistency, flow, and the use of colors and icons. Additionally, such resources as the OSF/Motif Style Guide and the Common User Access (CUA) guideline from IBM have served as foundations for the construction of the SLATE User Interface. The color and icon selection has proven to be the hardest problem to solve, and we are still working in these areas to try and maintain simplicity and consistency.

## Text or Graphics/Horizontal or Vertical?

SLATE provides access to a common design database for a systems engineering team to develop, analyze and optimize as needed. At TI, a systems engineering team is made up of a cross discipline of engineers that work together to develop a product in a concurrent engineering environment. Thus the reliability, producibility, safety, test (etc.) and design engineers all look at a common database. Our analysis showed that they all looked at the data from a different perspectives and converged towards an optimum solution that was called "the system". SLATE had to provide the ability to view the data from multiple perspectives to accommodate this user scenario.

In trying to provide multiple views, some people liked to work with text, some with graphics, some with graphics and text and of course some didn't like to work at all, but we had to come to a conclusion on what we could provide for the first release and this has wound up being all the above. The next problem was how to display the data in a window, horizontally or vertically?

Since a lot of data was hierarchical in nature, two views were developed for hierarchical data input. The Outline View was designed around an outliner paradigm that shows the hierarchical relationships of objects by indention from left to right and top to bottom, vertically down the screen. Since this does not satisfy everyone, a graphical Tree View was designed to represent the same data using graphic symbols and parent child links. The User Interface allows the user to chose their preferred view and switch between them. The Outline View tends to be preferred to see more data on the screen, while the Tree View tends to be used for printing and entering data by users who talk in terms of block diagrams.

The amount of data that can be attached to objects in the database, and the orientation of the data on the computer screen poses quite a difficult User Interface problem to solve. SLATE has chosen to provide a vertical outline paradigm that expands horizontally as complex hierarchies are traversed in the Outline View. The graphical hierarchical views

are top-down oriented and flow from top to bottom. A future release will provide horizontal orientation of the hierarchical trees in addition to top-down they will be left to right and the user will be able to select the preference of orientation.

## What About Those Icons?

Icons can serve to express large amounts of information concisely, provided that the icon meaning is readily discernible to the user. A well-designed set of icons can truly complement a user interface; poorly designed, non-intuitive icons simply results in a garish User Interface. In the SLATE User Interface, icons identify objects, identify object relationships, and object action commands available to the user. When selected they will open the appropriate window for the object chosen and present the data attached to the base object in the new window. The initial set of icons did not necessarily lend itself to being intuitive or definable at a glance and often led to confusion as to the meaning of the icon.

A small number of the icons used within the SLATE User Interface correspond directly to icons used in applications throughout the industry; in these cases, the meaning is presumably well-understood. However, for the situations in which icons were needed and there was no well-understood industry standard a "placeholder" icon was used while alternatives were considered. Often, icons had to be considered in groups, such as the pair of icons that reflect both directions of a "Defines" or a "Complies with" relationship. Where possible, guidelines were developed for each of the types of icons listed previously:

- o For relationship icons, shading was used to indicate the selected object's role in the relationship.
- o Object icons were given a 3-dimensional appearance through the use of coloring and shadows.
- o Icons for the various "create" commands were given a solid white background.

The new icons were exposed to users and revised based on user feedback; much confusion was generated over the icons depicting various relationships and traceability mapping. The addition of a help screen showing all of the SLATE icons and their meanings has proved invaluable.

## Those !*&%$ Colors

Why would colors be a problem? The user wants extensive use of colors and they must be user definable. This is great for 86% of the population, but what about the 14% that are partially or totally color blind, or work at monochrome worstations? This requires the use of colors and visual change of the objects in order to assist everyone to see the actions being taken on the selected objects. Color selection is equally a matter of science (what color combinations work well for displaying detailed information), art (what colors are aesthetically pleasing), and personal taste (what colors are consistent with the user's workspace).

In some cases, there are user expectations as to color usage. From the SLATE prototype, certain color usage (objects displayed in red) indicated whether an object could be modified or not; this usage was carried over into the SLATE product, both on the object level and in the support of symbolically referencing values from other objects in the database.

Over the course of SLATE development, the significance of color selection became readily apparent. The early versions of the User Interface did not support user-definable selection of colors in a manner considered anywhere close to easy. Experimentally, an initial set of color combinations was created which was acceptable for everyday use and demonstration. However, a frequent comment heard during early product demonstrations was that the default colors did not satisfy the user's individual tastes. Fortunately, later versions of the User Interface have addressed the color selection problem in a more flexible way; now colors can be defined by the user much like for other X Window-based applications.

## What Do Those &%$#** Arrows Mean?

Initially the traceability relationship between objects was represented by an arrow. The thought was that the arrow would point towards the destination of the link. An arrow point from a requirement to a complying system element represented a "Defines" link at least to some people, other thought it pointed at the complying object and thus must be a "Complying" link. Therein lies the problem. No matter how the designer defined it, it was interpreted by the logic of the person viewing the data. It became the source of many heated discussions as to what those &%$#** arrows meant.

We discovered that the traceability link was indeed a two-way relationship through the use of the arrows. The interpretation depended on the perspective of the user and which way the user was thinking the relationship flowed. So extensive input from users was solicited and with the ingenuity of one of our design engineers a new set of trace link Icons was developed that has passed the acid test of user acceptance. That means we have explained what they mean

during training class and we don't get any subsequent phone calls trying to settle an argument as to their meaning. We have also put an on-line help screen in the User Interface that defines all of the system Icons.

## In Retrospect, I.E. A Summary

The SLATE program had developed a prototype of the SLATE paradigm and as a result of the feedback on the work flow and interface, the productionization phase employed a human factors expert from the beginning to design the interface to industry standards. This worked well until the project human factors engineer left TI. The replacement spent 8 months trying to figure out what was up, and then added his flavor of interfacing to the design and we could not reach closure with the users or our system engineer. User Interfaces need user input early which means prototyping and usability testing during the entire design.

We found that what seemed logical to annotate in the database for trace links and attributes, was extremely difficult to represent on a computer interface in a manner that reduces the amount of data being viewed as well as conveys the message intuitively to the user. This problem has not been totally solved yet in light of the many link types that are foreseen for the next generation traceability capability. We seem to be making headway, but we need to be much more creative in the next level of information to be added to the SLATE Windows and Views.

## Current Status and Future Plans

SLATE is currently in production release and requirements are being analyzed for the next interim and major releases. Texas Instruments has licensed the SLATE technology, distribution rights and development roadmap to TD Technologies Incorporated of Cleveland, Ohio. TD Technologies has opened a SLATE division in Richardson, Texas to further the research and development of SLATE as well as the support and training of SLATE customers. It is our intention to continue developing and maturing the implementation of the Engineering of Complex Systems Technology Program Requirements Traceability Models. They have become the center of our Requirements Management capability. We also hope to incorporate other technology being developed as a result of the ECS program research into the SLATE environment to further improve the capabilities of Requirement Management for engineers designing complex systems.

# DOORS to the Digitized Battlefield: Managing Requirements Discovery and Traceability

Nancy Rundlet
Zycad Corporation
100 Enterprise Drive, Suite 500
Rockaway, NJ 07866

William D. Miller
AT&T Bell Laboratories
67 Whippany Road, P.O. Box 903
Whippany, NJ 07981-0903

**Abstract.** The integration of systems to create a stable, robust *system of systems* poses challenges to systems engineering management when the individual systems already exist, are numerous and rapidly evolving. The Dynamic Object Oriented Requirements System (DOORS™) introduces advanced object-oriented techniques to manage the complexity of large numbers of inter-linked requirements. DOORS provides an easy-to-use, on-line environment. Different on-line views of the project status are available in both text and hierarchical graphical formats by filtering and sorting on the attributes of the objects. This object-oriented approach to requirements management is extended to provide a *groupware* systems engineering management environment in support of the U.S. Army's program to *digitize* the battlefield of the future. The Army's intent is to improve their comparative advantage in combat effectiveness over that of potential adversaries. The magnitude and complexity of the effort to manage the project compared to classical requirements management methodologies and tools are discussed.

## INTRODUCTION

**Digitization of the Battlefield.** The U. S. Army has embarked on an ambitious concept to provide a comparative advantage in combat effectiveness over that of potential adversaries in any future conflict. This is to be achieved by sharing information through data-linking command posts with individual weapon platforms such as tanks, helicopters, and artillery cannon. The *digitization* of the battlefield will enable the following time-critical capabilities that improve combat effectiveness in contrast to current methods that use limited voice communications and manual transfer of information (Ross 1994):

- share situational awareness of terrain, friendly forces, and opposing forces
- provide a common view of the battlefield
- enable horizontal communications among disparate, distributed platforms to reduce operational timelines to synchronize maneuver and facilitate target handover.

The concept is illustrated in Figure 1.

The *digitization* architecture initiative is a multi-year effort managed using an iterative, evolutionary approach with several phases in order to reduce the risk of such a complex undertaking. In one dimension of complexity, the project requires interfacing with tens of other contractor or government laboratory teams, each with evolving user requirements, system requirements, and separate milestones for fielding of prototypes or operational capabilities. The number of external interfaces and the nature of their dynamics poses significant systems engineering management challenges to successfully integrate this *systems of systems.*

**Requirements Management.** The use of requirements management tools integrating *easy-to-use* Graphical User Interfaces (GUIs) promoting *visual awareness* of project/engineering data, the emerging technology of *object-oriented* databases that are an order-of-magnitude faster than relational

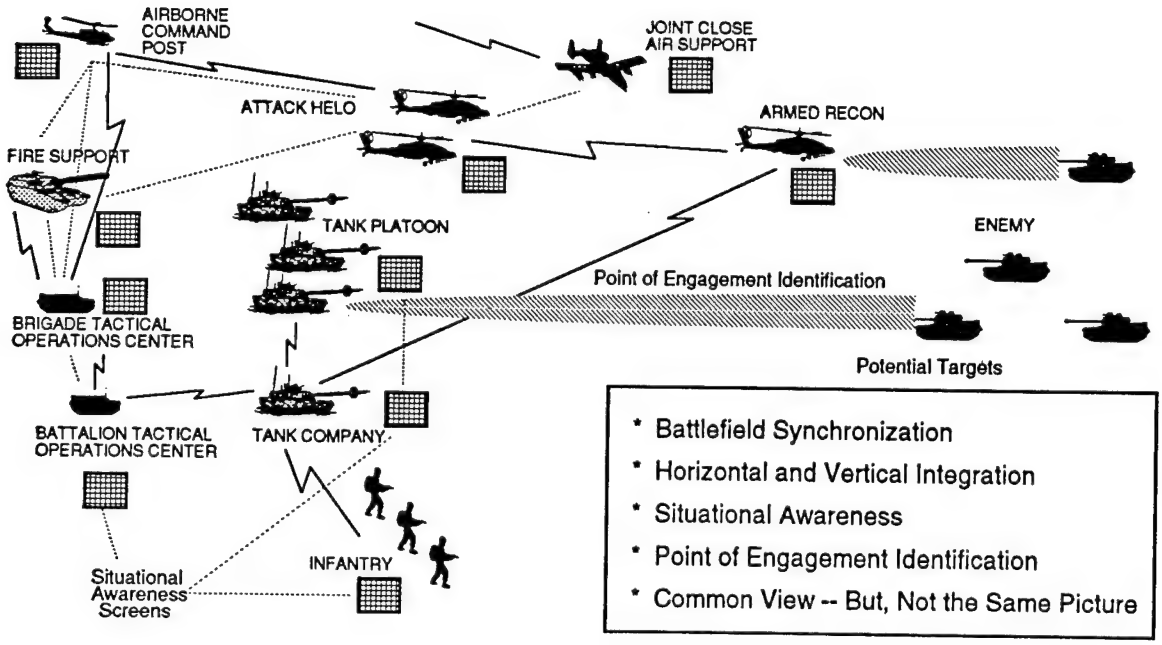



**Figure 1.** The digitization of the battlefield provides situational awareness, common battlefield view, and horizontal communications to combat platforms and command posts.

databases (Atwood 1991), and *networkability* to connect distributed/dispersed project teams provides program managers and systems engineers a quantum improvement in capability to manage the enormous complexity of the project.

The Dynamic Object Oriented Requirements System (DOORS™) provides such capabilities to more completely manage programs such as the digitization of the battlefield project. DOORS is a general purpose system based on well known, *object-oriented* principles for organizing information, i.e., hierarchy, abstraction, characterization, polymorphism, reuse, and the relationships between sets. This produces a *methodless* way of structuring masses of information, yet tailorable to organizations' standard processes. Requirements are objects with attachable attributes. Objects can be linked and the linkages can also have attachable attributes. DOORS provides a set of predefined attributes for requirements management (QSS 1993) such as :

| attribute | type |
|---|---|
| created by | string |
| created on | date |
| last modified by | string |
| last modified on | date |
| object text | text |
| object short text | string |
| created thru | enumeration |

DOORS provides capabilities to import and parse massive amounts of existing information without the need for manual reentry. It also provides on-line ad-hoc queries, filtering, and sorting as well as export capabilities for publishing documents. A DOORS Extension Language (DXL) provides the capability to tailor the tool to organizations' standard processes and reports. An Applications Programming Interface (API) provides interoperability to other tools and software applications.

## REQUIREMENTS/SPECIFICATION FRAMEWORK

Traceability is established from user requirements provided by the Army to the system requirements derived by the contractor. The system requirements are linked to the digitization architecture which, in turn, are linked to the architecture element functional specifications.

The progress of the digitization project is assessed using exit criteria for Situation Awareness, Command and Control on the Move, Seamless Communications, and Battlefield Synchronization. Minimum and goal oriented technical and operational performance metrics are defined for the exit criteria and are benchmarked against the current baseline capabilities/performance. The exit criteria

are assessed using a combination of modeling, simulation and demonstrations.

Verification of the requirements/specifications is achieved bottom up by first verifying the architecture element specifications by tests/demonstrations of the architecture element prototypes. The digitization architecture is verified through system performance modeling and simulation of the interoperability of the elements. The system requirements are verified by the use of operational warfighting models and simulations. The user requirements are then verified by conducting interactive simulations and integrated prototype demonstrations.

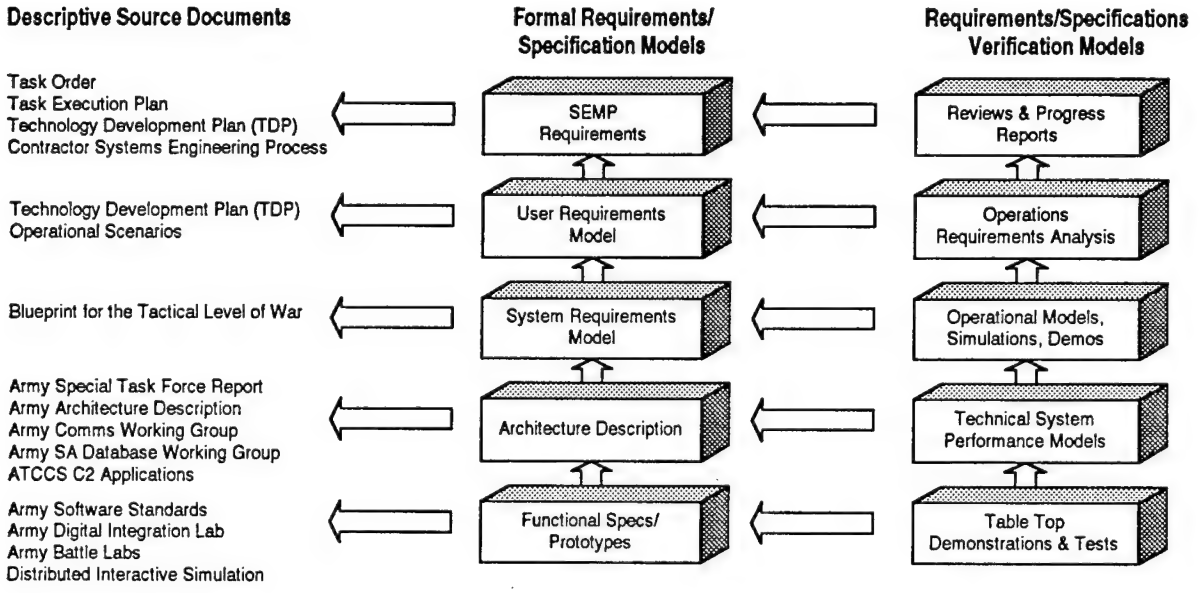



**Figure 2.** The requirements/specification framework establishes the traceability between different levels of descriptive source documents, requirements/specifications and their verification.

Overlaying the requirements/specification and verification process is a project management process using draft Mil-Std-499B, Systems Engineering Management, as guidance. Figure 2 shows the framework to manage and control the program.

**User Requirements.** *User requirements* define the problem or results to be produced for the user whereas the *system requirements*, discussed in the next section, define the abstract solution in response to the user requirements. User requirements are stated in terms of the following types of objects (Stevens 1993a):

- types of users,
- user capabilities
- user constraints
- user risks
- user test/verification requirements.

The digitization project has identified several hundred unique user types. Examples are Brigade Commander, Armored Battalion Tactical Command Post, Scout Dismounted, Tank Platoon Leader, Mortar Squad, Ambulance, Engineer Battalion Trains, etc. Each user type has a unique identification code, a rule, type description, computer types, and radio types.

The capabilities define the results provided to the users, hierarchically organized through time. The following attributes are examples of those defined in DOORS for each of the capability object types, in addition to the predefined attributes used for *requirements management*:

| capability attributes | type |
|---|---|
| need | string |
| priority | enumeration |
| stability | string |
| verifiability | Boolean |
| issues | text |
| performance goal | integer or real. |

Linkages with attributes are established as shown in Figure 3 between the user types, capabilities, and constraints. These linkages allow multidimensional views of the user requirements.

**System Requirements.** System requirements are derived using hierarchical object-oriented analysis and linked back to the user requirements. The system requirements model includes the following:

- object functionality
- internal interfaces
- dynamics/behavior
- performance
- external systems requirements
- risks
- specialty engineering constraints
- verification requirements.

An abstraction of the system requirements model is shown on the right hand side in Figure 4. This model is different from the user requirements but is traceable to them as shown by the links between the user requirements and system requirements in the Figure.

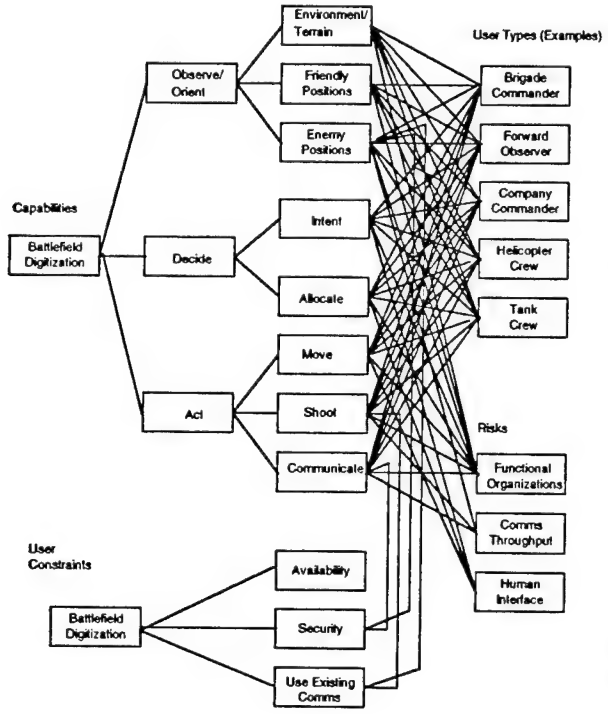



**Figure 3.** The user requirements consist of linked models of user capabilities, user constraints, user types, and user risks.

**Architecture.** The system architecture is synthesized and linked back to the system requirements. The architecture model consists of identified architectural elements, performance capabilities of the elements, constraints on the elements, risks, and verification methods for the elements as shown in Figure 5.

**Architectural Element Functional Specifications/ Prototypes.** Specifications and prototypes are to be provided for the major architecture elements: Soldier-Machine-Interface (SMI), SA database, $C^2$ applications, information structure/database and communications. The specifications define the functionality that meet the digitization project exit criteria. The functional specifications are augmented by working prototypes demonstrable in a laboratory environment. The prototypes are evolved using a spiral approach of *build a little, test a little* to evaluate the effectiveness of the architecture elements and measure the satisfaction of user needs. This approach is used given the knowledge that user requirements are not fully defined and that user feedback will impact the stability of the user requirements, as well as the derived system requirements and architecture description.

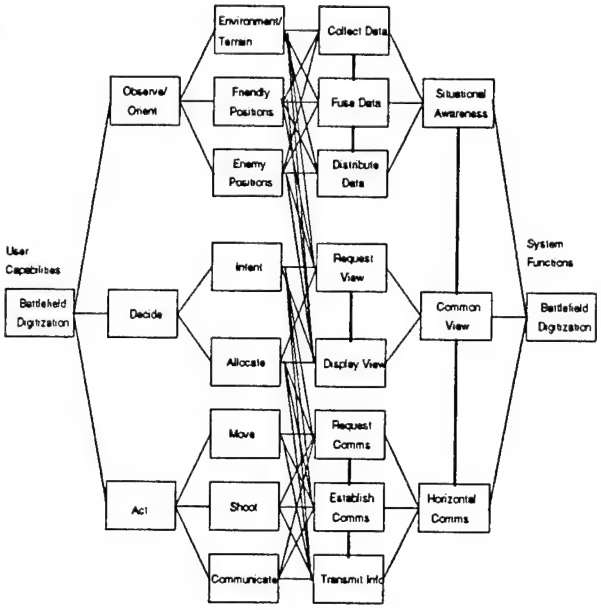



**Figure 4.** The user requirements and system requirements are separate, linked models.

## PROJECT/SYSTEMS ENGINEERING MANAGEMENT REQUIREMENTS

The digitization project is being managed as an evolutionary project with milestones for user requirements, system requirements, architecture, simulation/demonstration verification in each phase. The evolution is obtained through a framework architecture which attempts to define uniform, stable *system of system* interfaces. The framework architecture separates those things which are stable

from those which will change. All modules can be tested individually against a standard interface. The interface can be upgraded as long as changes are upward compatible. Figure 5 also shows the project management requirements for Configuration Items (CIs) and schedule which together constitute the project milestones.

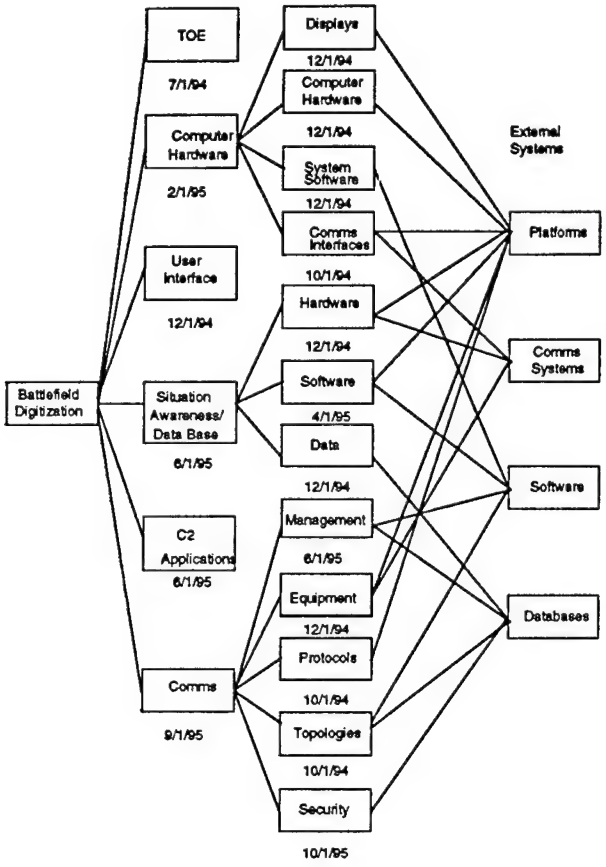



**Figure 5.** The architecture model includes both internal and external elements and the interfaces between them. The architecture model is further linked back to the system requirements model and linked forward to the architectural elements (not shown in figure).

**Requirements for Requirements Management.** The requirements for requirements management (Stevens 1993b) are stated and linked to relevant project team requirements for managing the project. A summary of some key performance requirements is shown in Table 1.

The set of *requirements for requirements management* constrains the supporting environment and tools to be usable by non-dedicated, non-specialists. This improves the performance of the information flow or circulation in a project in contrast to the inertia experienced when tools are perceived as hard-to-use and are attended to by a few dedicated specialists on a project with whom most of the project team members must interact to get the information. In particular, managers should have easy access and be able to successfully navigate tools to get project information with the same comfort as using a spreadsheet application. One aspect of ease of access is that the tool should be readily available to casual users at their desktops without requiring a high cost workstation. Client/server computing with X-Windows terminals or personal computer emulation software provides such an economical environment without having to equip everyone with a high-cost workstation.

**Table 1.** Summary of requirements management performance for the project.

| Requirement | Performance |
|---|---|
| Managers training | 1 day maximum |
| General users training | 2 days maximum |
| System administration set up time | less than 1 hour |
| Electronically import requirements from external database | 1 day technical assistance maximum |
| User time to tailor a publishing template | less than 10 minutes |
| Users document navigation training | less than 3 minutes |
| Object/link attribute definition training | less than 5 minutes |
| Number of requirements | at least 20,000 |
| Levels of hierarchy | 20 levels |
| Standard query response time | within 0.25 seconds |
| Filtering/sort operation on 1,000 requirements | within 5 seconds |

## CONCLUSIONS

Traditional approaches to requirements management break down for iterative or evolutionary concept evaluation projects that depend upon many parallel projects. An object-oriented approach augmented with multiple hierarchies with linkages achieves the traceability of an otherwise intractable problem. The use of an automated tool such as DOORS has been successful in facilitating this project. The process, methodology, and environment have been put in-place to achieve a new capability that integrates users, program managers, project managers, systems engineers, and design teams.

## REFERENCES


Atwood, Thomas M., Object Design Inc., *The case for object-oriented databases,* IEEE Spectrum, February 1991, pp. 44-47.

Quality Systems and Software Limited (QSS), *DOORS 2.0 Reference Manual,* 1993.

Ross, Gen. Jimmy D., *Winning the Information War,* Army, Vol. 44, No. 2, February 1994, pp. 26-32.

Stevens, Richard, *Structured Requirements,* QSS Ltd, Oxford, UK, 1993.

Stevens, Richard, *Requirements for Requirements Management,* QSS Ltd, Oxford, UK, 1993.

Tiel, Richard F., *Using Structured Process Improvement Techniques on the Systems Engineering Process,* Proceedings of the Third Annual International Symposium of the National Council on Systems Engineering, July 26-28, 1993, Arlington, VA, pp. 301-307.


## AUTHORS' BIOGRAPHIES

**Nancy Rundlet** is Director of Sales for Requirements Management at Zycad Corporation in Rockaway, New Jersey. She is responsible for managing the sales and marketing of Requirements Management solutions. Mrs. Rundlet has been in the field of engineering for 18 years, including two years at Zycad, 14 years at Hewlett-Packard, and two years at General Cable Corporation. She is a member of the National Council on Systems Engineering (NCOSE) and is the President of the Liberty Chapter for New Jersey, New York, and Pennsylvania. Mrs. Rundlet received her Bachelors Degree in Physics and Mathematics from Gettysburg College and her Masters in Engineering Management from Stevens Institute of Technology.

**William D. Miller** is Technical Manager of a Systems Engineering Group at AT&T Bell Laboratories in Whippany, New Jersey. He joined AT&T Bell Laboratories in 1973 and has 21 years professional experience as a systems engineer and product manager in specialized communications networks for both commercial and military applications. Mr. Miller is a member of AFCEA, IEEE, and NCOSE. He is Secretary of the NCOSE Liberty Chapter for New Jersey, New York, and Pennsylvania. Mr. Miller received his BSEE in 1971 and his MSEE in 1973, both from the Pennsylvania State University.

**The views expressed in this paper are those of the authors and do not reflect the official policy or position of the U. S. Army, the Department of Defense, or the U. S. Government.**

TITLE: Formal Requirements Specification Languages for Real-Time Systems: Expressibility Issues

AUTHOR: Ralph D. Jeffords

AREA OF INTEREST: Formal Requirements Specification

ADDRESS:

Ralph D. Jeffords
Code 5546
Naval Research Laboratory
4555 Overlook Ave., SW
Washington DC 20375

jeffords@itd.nrl.navy.mil

PHONE: (202) 404-8493

FAX: (202) 404-7942

# Formal Requirements Specification Languages for Real-Time Systems: Expressibility Issues

R. D. Jeffords*

Abstract

A major conclusion of our study comparing different formal specification languages for real-time systems is that these languages are lacking in expressibility. We identify two contributing factors: (1) too little emphasis on the environment and (2) too much emphasis on verification. To remedy the situation we advocate a dual approach to formal specification of real-time systems: (1) statement of Critical System Properties and (2) specification of a System Model relating the system to its environment. Formal verification of the System Model with respect to the Critical System Properties increases confidence in the the system requirements specification. In support of the dual approach, we discuss a number of general expressibility issues: the need for asynchronous communication, equal treatment of states and events, and appropriate level of abstraction in support of families of requirements, and several issues particular to real-time: dense (continuous) time models and availability of constructs for both events and actions. We conclude that although no single language will have all the desired features, language designers should strive for a reasonable compromise between expressibility and verification.

**Keywords:** formal requirements specification, verification, expressibility, real-time systems.

## 1 Introduction

Over the past year, we have compared of a number of different formal specification languages and tools for developing real-time systems. In the study, we used a benchmark problem, the Generalized Railroad Crossing, which extends the example given by Leveson and Stolzy [17]. Our approach has been to examine solutions to this problem obtained from experts in using the method or tool, to perform additional experiments with the method or tool, and to report upon our experience [7, 8, 9].

One of our major conclusions is that current specification languages for representing real-time behavior are lacking in *expressibility*, i.e., availability of appropriate constructs for developing a requirements specification in a straightforward manner and for making that specification readable. We have found, in particular, that it may be difficult to express precisely what is intended without resorting to artifacts and restrictions that detract from the overall understanding of the formal requirements specification.

We have identified two factors that contribute to this lack of expressibility: (1) too little emphasis on specifying the system environment, and (2) too much emphasis on ease of verification.

### 1.1 Lack of Expressibility of the Environment

Lack of expressibility of the system environment is due in part to the fact that formal requirements specification has its roots in description of software systems, which have fewer and simpler environmental interactions than most embedded real-time systems. For example, process algebras, such as CSP (Communicating Sequential Processes) [12] and CCS (Calculus of Communicating Systems) [19], emphasize synchronous interaction, which is not an appropriate model for the uncontrollable part of the system environment which is inherently asynchronous.

*R. D. Jeffords is with the Naval Research Laboratory, Washington, DC 20375. The work reported here is supported in part by a grant from the Naval Surface Warfare Center.

Furthermore the environment often represents fixed constraints upon the requirements–such as physical laws–while the system to be built may allow many different implementations. In such situations the requirements language should allow a description of these physical laws separate from the description of the system. Parnas and Madey provide such a separation through relations *NAT* ("NATural" or uncontrollable environment) and *REQ* (remaining "REQuirements" of the system) [22].

### 1.2 Too Much Emphasis upon Verification

Ease of formal verification is an important concern for formal methods but should not be at the expense of expressibility. For example, to simplify verification of real-time systems it is easier to consider only discrete time (or even no timing model), thus limiting the expression of the system's timing properties. What is desirable, but may be difficult to achieve, is a smooth transition between simplified models–which may permit easier verification and understanding–and more complete models. By smooth transition we mean formally showing that the results for simplified models are a special case of the results for more complete models, and that the analysis of simplified models provides useful feedback for correcting errors in the requirements. Methodology for such smooth transition is largely lacking: formal languages intended for simplified models usually provide little guidance in how simplified models relate to more complete models.

We do not deny the importance of verification. We firmly believe that verification of critical real-time properties, e.g., analysis based upon a mathematical proof, are invaluable at the requirements level to avoid costly changes to the requirements later in the system development life cycle. System simulation (including "executing the requirements," and system prototyping), although useful for understanding requirements, is not usually sufficient to establish such critical properties, unless it is exhaustive. Verification, on the other hand, does provide the equivalent of such exhaustive simulation and may be more cost-effective than exhaustive simulation.

### 1.3 Overview

This paper recommends a formal development method that represents the requirements in two parts. Environmental concerns are manifest in a set of Critical System Properties that must be guaranteed for safe, dependable behavior of the system. Behavior of the system is represented by an operational System Model. Verification of the System Model with respect to the Critical System Properties provides for the early detection of system errors.

We also outline our position on a number of issues related to the expressibility of languages for real-time systems–namely, asynchronous communication, dense time, and level of abstraction in requirements specifications. We also indicate some useful modeling techniques and some precautions when more expressive constructs are not available.

## 2 Structuring Real-Time Requirements Specifications

We advocate the partition of real-time system requirements into two parts to maintain a balance between environmental concerns and requirements level verification:

- Critical System Properties: These are the required relationships among environmental entities to include both those under system control (some aspects of Parnas and Madey *REQ*) as well as environmental assumptions (*NAT*). These are often safety properties, i.e., requirements that the system never reach *unsafe* states. For example, "if a train is in the crossing then the crossing gate must be down." Formalisms for stating properties should be easily translatable from such informal statements to aid in their validation.

- System Model: The System Model encompasses the required relationships between the system and its environment (more detailed aspects of *REQ*). This is often an operational model, such as a finite state machine, which describes the behavior of the system. Care must be taken

that such models represent only the observable behavior and do not inject premature design decisions ("how" rather than "what" ).

Even though some would consider such requirements, in particular the system model, to be overspecified, we feel that this dual approach gives a greater confidence in the validity of the requirements, especially when we can formally verify that the system model satisfies those critical properties.

A refinement of this method is currently under investigation at NRL. This additionally includes system simulation for customer validation of requirements and detailed consistency and completeness checks related to a formal definition of the systems model. Progress to date includes development of the formal semantics (and required consistency and completeness checks) for a variant of the SCR requirements specification technique as the System Modeling language, and the development of a prototype toolset comprised of specification editor, simulator, and consistency and completeness checker [10, 11]. Future work includes more sophisticated consistency and completeness checking and verification of critical properties using currently available general purpose theorem proving systems or model-checkers.

# 3 Expressibility Concerns

Below, we present several expressibility issues. Expressibility and other desirable criteria for real-time requirements methods are outlined in [3]. Examples based on the methods motivate many of the opinions expressed in this paper and are presented in [7].

Our views are based largely on our experiences with the following specification languages and tools: the process algebra tools FDR (Failures Divergence Refinement) [5] for CSP models [12], the Concurrency Workbench [2] for (temporal) CCS models [19, 20], VERSA for ACSR (Algebra of Communicating Shared Resources) [16]; the Modechart verification system [14]; automated general-purpose specification and verification systems PVS (Prototype Verification System) [21, 23] and EVES (Environment for the Verification and Evaluation of Systems) [4, 15]; Lynch-Vaandrager timed automata [18]; and ASTRAL [6].

## 3.1 Expressibility in Untimed Models

We first address some general issues about specifying concurrent systems that are largely independent of real-time concerns.

### 3.1.1 State-Based vs. Event-Based

*POSITION: States and events should both be first-class citizens, i.e. there should not be overemphasis of one to the extent that it is difficult to express the other.*

State transition system models generally treat states and events equally. Other methods, such as process algebras and RTL (Real-Time Logic) [13, 14] are event-based. The difficulty with event-based languages is that it may be difficult to express properties, such as safety properties, that are most easily described with reference to system state, and thus difficult to validate that the formal statement captures the intended meaning. For example, it is quite easy to translate the informal statement, "if a train is in the crossing then the gate must be down," to the formal statement using state: "State-of-Crossing = Some-train-in-crossing implies State-of-Gate = Gate-Down."

We have noted two techniques that alleviate some of the difficulties of expressing state in the System Properties of event-based models:

- The state in a process algebra can be captured explicitly by a self-looping transition on an artificial event. Detecting that a transition is possible on this artificial event is equivalent to testing that the system is in that state [1].

- Parameterization of processes in the extended CSP notation used by FDR [5] is analogous to Alan Shaw's Communicating Real-Time State Machines [24]. This notation provides a powerful means of compact description with state information represented by the parameters.

### 3.1.2 Interaction: Synchronous vs. Asynchronous

*POSITION: The specification language should support both synchronous (preferably multi-way) and asynchronous communication.*

Because parts of the environment are outside the system's control, some environmental quantities must be described as asynchronous inputs. Asynchronous events can be modeled via synchronous rendezvous, but doing so requires that the receiver of such events can never be blocked (i.e., the receiver can never delay in responding to the rendezvous).

### 3.1.3 Declarative vs. Operational Properties

*POSITION: If possible, avoid operational specification of properties.*

Several tools that we have surveyed use largely operational notations for system properties. Such notations sacrifice the understandability of straightforward declarative properties, as well as pose difficulty in validation due to translation from an often natural declarative form to a radically different operational form:

- To expedite verification, FDR limits specification of properties to the same operational process models used for specifying the system.
- The Concurrency Workbench, which provides for analysis of (temporal) CCS models, contrasts with FDR in allowing declarative forms of properties in conjunction with model-checking [2].

### 3.1.4 Concrete vs. Abstract Specifications

*POSITION: The specification language should provide the appropriate level of abstraction for describing requirements of families of related systems. It should be a secondary concern to use concrete approximations (to include discrete time) for the sake of tool-assisted verification, and to validate the concrete model and its analysis versus the abstract model.*

We have noted that completely automated verification tools appropriate to real-time system verification require quite stringent simplifications of the requirements model: finite state systems, discrete time model, concrete values for timing and other parameters with only implicit relationships among these parameters [8, 9]. On the other hand, specification and verification via general purpose theorem proving systems, such as PVS [21, 23] and EVES [4, 15], allows infinite state systems, dense timing models, and symbolic values for timing and other parameters with explicit relationships among these parameters. This higher level abstraction provides for (a) better understanding of the boundary conditions of the system (due to explicit boundary parameter relationships), (b) reuse over a wider range, and (c) often more concise (and thus clearer) specifications.

## 3.2 Expressibility in Timed Models

We next outline various issues related to extending concurrent systems with timing properties.

### 3.2.1 Actions vs. Events

*POSITION: The language should provide for both* actions, *which require finite time to complete, as well as* events, *which are considered to be instantaneous.*

The problem is with languages which provide only actions and not events, such as ASTRAL[6]. It is relatively simple to model actions via distinct "start action" and "complete action" events,

but conversely it difficult to model events via actions since this requires artificial spacing of events that should be considered simultaneous. It may argued that such spacing of events will cause no problems if this artificial spacing is chosen small enough, but this seems an unnecessary complication compared to saying that such events are simultaneous.

### 3.2.2 Discrete vs. Dense Time

*POSITION: The specifier should be able to state the timing requirements precisely via a dense time model, i.e. one in which there are an infinite number of time points within any interval such as the rational numbers or the real numbers.*

A number of verification tools use a discrete time model to allow for completely automated verification. The problem is that this may introduce error in the specification if one does not validate formally that the discrete analysis can be extended to the more general case. There are two ways of performing such a discretization for events occurring between two clock ticks:

- All events occurring between ticks $i$ and $i+1$ are modeled as occurring at time $i$.
- For each event e occurring between ticks $i$ and $i+1$ it is known that $i \leq time(e) \leq i+1$, and optionally the ordering of some of these events may be known.

The former approach–used by VERSA, the Concurrency Workbench, and Modechart–is probably easier since it deals with a single point in time for each event, as opposed to time intervals associated with each event in the latter approach. The important consideration, which is usually not emphasized, is how to validate discrete time models with respect to the actual system.

Discretization of time in event-based modeling with synchronous communication, such as process algebras, where the clock tick is simply another event requires caution. One must ensure that the clock remains "live," i.e., there will always be only some finite number of events after each clock tick until the occurrence of the next tick.

### 3.2.3 Basic Timing Constructs for Hard Real-Time Systems

*POSITION: Basic timing for hard real-time systems should include both upper bounds (deadlines) and lower bounds (delays) upon the occurrence of events with respect to appropriate reference points.*

Such interval timings provide appropriate basis for analysis while providing sufficient generality to account for limited unknown behavior of system components. Timing for actions can easily be modeled with this construct. It is useful to express timings over large parts of the systems, such as at levels larger than single transitions of a state-transition system, however such expression may not be amenable to graphical notations as with single transition timings: simply label the arc that corresponds to the transition.

### 3.2.4 Macro-Steps vs. Micro-Steps

*POSITION: The specifier should be able to choose between specification and analysis based upon observable* macro-steps *as well as hidden* micro-steps.

By macro-steps we mean observable behavior that represents a state which remains unchanged for some non-zero time. Micro-steps correspond to changes that occur (nearly) simultaneously so are not observable. Languages which strictly interleave events or actions usually treat each such event or action as a micro-step. Languages which treat a number of events or actions truly occurring in parallel may not have such a subdivision into micro-steps (although tools based upon the language will often require microsteps for the implementation of the language).

For some types of analyses all micro-steps at the same time may be important since they correspond to system behavior. In other situations such micro-steps may be irrelevant, or due to

characteristics of a modeling technique may be completely artificial. Visibility of all micro-steps is also useful for debugging specifications developed under tool support.

For example, in the formal SCR System Modeling language under development at NRL, the micro-steps may be considered an artifact of the specification technique. The micro-steps are just a device for ensuring consistent definition of each macro-step [10]. In this situation we only care about the state of the system at "equilibrium"–while waiting for another appropriate environmental input to the system.

One problem that we have noted with models that strictly interleave events when micro-steps are considered important is that a safety property such as "if a train is in the crossing then the gate must be down" requires the gate to be lowered *strictly* before any train arrives. This is because due to the interleaving and the independence of the train and gate events we do not know which event occurs first when simultaneous as we may see these two events interleaved in either order. This means safety properties are not "closed" in the sense that safety properties do not still hold in the limiting case. This absence of "closure" makes it more difficult to treat optimal solutions, e.g., the latest the gate could be lowered for the previous example.

#### 3.2.5 Hard vs. Soft Real-Time

*POSITION: Modelers should have a choice of both soft and hard real-time timing constructs within a uniform modeling environment.*

Most formal real-time techniques have concentrated on hard real-time: the system must satisfy its deadline and delay timing constraints or it fails. There is also a need to pay more attention to soft real-time concerns: less stringently, the system meets its timing constraints with some probability. Ideally one would want a choice of methods for both types of timing characteristics based upon a common underlying specification. At NRL we are investigating the feasibility of such an approach.

## 4 Summary and Conclusions

In this paper we have stated our position on a number expressibility concerns based upon what has been lacking or difficult to express in our experience:

- General modeling concerns: equal treatment of events and states, expression of both asynchronous and synchronous behavior, need for declarative form for Critical System Properties, concrete vs. abstract System Models.
- Timing concerns: actions vs. events, dense vs. discrete time, and micro-steps vs. macro-steps.

We conclude that a single language that satisfies each position outlined in this paper is probably not feasible. To cover these positions fully would require the complexity of many different and incompatible modeling and verification techniques, since these positions cover many, often conflicting alternatives and trade-offs. Furthermore research in formal real-time specification languages is still in its infancy, so there is still considerable debate over what features should be included in real-time specification languages.

We offer these opinions to remind language designers that there should be a reasonable compromise between the needs of the specifier to express requirements directly and naturally without resorting to artifices introduced for verification or other concerns.

## Acknowledgments

I thank my colleagues Constance Heitmeyer and Bruce Labaw of the Naval Research Laboratory for their suggestions for improving this paper.

# References


[1] R. Cleaveland, Personal Communication, Aug. 1993.

[2] R. Cleaveland, J. Parrow, and B. Steffen, "The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems," *ACM Trans. Prog. Lang. and Sys.*, Vol. 15, No. 1, Jan. 1993, pp. 36-72.

[3] P. C. Clements, C. E. Gasarch, and R. D. Jeffords, "Evaluation Criteria for Real-Time Specification Languages," NRL Memo. Report 6935, Naval Research Lab., Wash., DC, 1992.

[4] D. H. Craigen, S. Kromodimoeljo, I. Meisels, B. Pase, and M. Saaltink, "EVES: An Overview," Proc. VDM '91, Noordwijkerhout, Netherlands, Springer-Verlag, New York, NY, Oct. 1991.

[5] "Failure Divergence Refinement, User Manual and Tutorial," Version 1.4 Formal Systems (Europe) Ltd., Oxford, UK, Dec. 1994.

[6] C. Ghezzi and R. Kemmerer, "ASTRAL: An Assertion Language for Specifying Real-Time Systems, *Proc. 3rd European Softw. Engin. Conf.*, Milan, ITALY, Oct. 1991.

[7] C. Heitmeyer and R. Jeffords, "Formal Specification and Verification of Real-Time System Requirements: A Comparison Study," Draft Tech. Report, Naval Research Lab., Wash., DC, Dec. 1993.

[8] C. Heitmeyer, R. Jeffords, B. Labaw, "A Benchmark for Comparing Different Approaches for Specifying and Verifying Real-Time Systems," *Proc., Tenth Intern. Workshop on Real-Time Operating Systems and Software*, May 13-14, 1993, New York, NY.

[9] C. Heitmeyer, R. Jeffords, B. Labaw, "Comparing Different Approaches for Specifying and Verifying Real-Time Systems," *Proc., 1993 CSESAW Workshop*, July 20-22, 1993, Wash., DC.

[10] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw, "SCR-Style Specification and Analysis of Requirements: A Formal Foundation," Draft Tech. Report, Naval Research Lab., Wash. DC, Nov. 1993.

[11] C. L. Heitmeyer and B. G. Labaw, "Consistency Checks for SCR-Style Requirements Specifications," NRL Report NRL/FR/5540–93-9586, Naval Research Lab., Wash., DC, Dec. 1993.

[12] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall Intl., Englewood Cliffs, NJ, 1985.

[13] F. Jahanian and A. K. Mok, "Safety Analysis of Timing Properties in Real-Time Systems," *IEEE Trans. Softw. Engin.* Vol. SE-12, No. 9 (Sept. 1986).

[14] F. Jahanian and D. A. Stuart, "A Method For Verifying Properties of Modechart Specifications," *Proc. Real-Time Systems Symposium*, Huntsville, AL, 6-8 Dec. 1988.

[15] S. Kromodimoeljo, W. Pase, M. Saaltink, D. Craigen, and I. Meisels, "A Tutorial on EVES," Odyssey Research Associates, Ottawa, Ont., Canada, 10 Feb. 1993.

[16] I. Lee, P. Brémonde-Grégoire, and R. Gerber, " A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems," Tech. Report MS-CIS-93-08, Dept. of Computer and Information Science, Univ. of PA, Jan. 1993.

[17] N. G. Leveson and J. L. Stolzy, "Analyzing Safety and Fault Tolerance Using Time Petri Nets," *TAPSOFT: Joint Conf. on Theory and Practice of Software Development,* Springer-Verlag, March 1985.

[18] N. Lynch, "Simulation Techniques for Proving Properties of Real-Time Systems," *Proc. REX Workshop '93, Lecture Notes in Computer Science*, Mook, NETHERLANDS, Springer-Verlag, New York, NY, (to appear).

[19] R. Milner, *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[20] A. Moller and C. Tofts, " A Temporal Calculus of Communicating Systems," *Proc. CONCUR'90, Lecture Notes in Computer Science 458*, Springer-Verlag, New York, NY, 1990, pp. 401-415.

[21] S. Owre, N. Shankar, and J. M. Rushby, "The PVS Specification Language (Beta Release)," Computer Science Lab., SRI Intl., Menlo Park, CA, Feb. 1993.

[22] D. L. Parnas and J. Madey, "Functional Documentation for Computer Systems Engineering (Version 2)," CRL Tech. Report 237, McMaster Univ., Hamilton, Ont., CANADA, TRIO (Telecommunications Research Inst. of Ontario), Sept. 1991.

[23] N. Shankar, "Verification of Real-Time Systems Using PVS," *Computer Aided Verification (CAV '93) LNCS 697*, Springer-Verlag, New York, pp. 280-291.

[24] A. C. Shaw, "Communicating Real-Time State Machines," *IEEE Trans. Softw. Engin.* Vol. 18, No. 9 (Sept. 1992), pp. 805-816.

# RECAP: A REquirements CAPture Tool for Large Complex Systems

Michelle M. Hugue
Michael Casey
Glenn Wood
Trident Systems, Inc.

Michael Edwards
NSWC

## Abstract

Complete, correct and consistent requirements are essential to the development of large complex systems. The RECAP REquirements CAPture prototype tool provides automated assistance in identifying, capturing, analyzing and using requirements. RECAP combines the information management and extraction capabilities of information retrieval system paradigms with knowledge-base rules to support efficient access to related requirements. RECAP does not replace the human's analytical skills, but enhances them. Sequential and string-search access to any portion of the document set is available Requirements physically separated in the document can be read and classified as if they were sequential. Quick access to information based on keywords, sentence identifiers, or on rule-based queries is also supported. The user is expected to provide some information to aid RECAP in resolving ambiguities, correcting mistakes, and adding missing terms or sentences. Similarly, RECAP will attempt to aid the user in making these decisions.The theoretical basis for RECAP is presented, and the baseline RECAP functions are described. Potential extensions to the current RECAP implementation are also explored.

# RECAP: A REquirements CAPture Tool for Large Complex Systems

Michelle M. Hugue Michael Casey
Michael Edwards Glenn Wood[1]

## 1 Introduction

Specifications for large complex systems, such as a Naval command and control system, an airspace battle management system or an international financial trading system, invariably consist of many documents containing pages full of jargon and often arcane symbols, which are meant to define the desired product fully. The requirements in these documents must be identified, interpreted, and usually extended to produce a design which, when instantiated as a product, satisfies the specifier. Too often, the product fails to meet such expectations because the supplied requirements were incomplete, inconsistent, incorrect or uninterpretable. A complete, correct, and precise set of requirements documents is essential to ensure that the desired system functionality is provided. Factors such as the inherent ambiguity of natural language representations, the scope and complexity of the target system applications, and the significant challenge of guaranteeing the dependability of the required system services combine to make writing, interpreting, and using requirements documents difficult.

As the human mind is limited in its ability to extract, assimilate, recall and transfer large volumes of information, the benefits of mechanical and digital assistance have been explored for years. Many techniques and tools have been proposed to address the complex research issues arising from the need for a robust text interpretation facility to support requirements elicitation and capture. Natural language understanding techniques abound, in the form of information retrieval systems, rule-based systems, and semantic analysis tools which attempt to extract meaning from the words, sentence formats and other context clues in a document [1] [2]. These techniques have been successful for small scale problems, enabling automatic construction of system dataflow and functional diagrams. However, scalability issues have limited the utility of these methods in dealing with the large sets of requirements that arise in many complex systems.

This paper presents an approach to requirements capture which addresses the scalability and complexity issues. Factors which can lead to the misinterpretation of requirements and prevent the implemented system from matching the original intentions such as forgotten requirements, extraneous requirements, and incorrect requirements are also addressed. Our approach combines information retrieval and rule-based systems theory to support requirements elicitation, capture, analysis, assessment, modification, and usage for large, complex systems. Information retrieval methods are used to organize large amounts of information. Common sense and domain-specific rules are applied to partition potential requirements into more manageable sets and to identify potential ambiguities, omissions, and inconsistencies in the requirements. The REquirements CAPture (RECAP) prototype tool being developed at Trident Systems, Inc., under NSWC oversight, embodies this approach by providing the user with multiple levels of automated support for requirements elicitation, capture, and analysis. RECAP is meant to augment the human analysts' skills, not to replace them. System developers and analysts can capture existing requirements from a specification, parse them into domain–specific groupings, and structure them into a various forms to support interactive analysis.

Before discussing RECAP operations, we briefly discuss techniques for extracting meaning from natural language text and present the theoretical basis for RECAP. Then, we describe the key features of RECAP, which fill needs not addressed by other tools. In the final section, we discuss the contributions of our work and further plans for developing RECAP.

## 2 Background

A major challenge in the automatic interpretation of natural language text is that all of the information needed to understand the meaning of a sentence is not contained therein. There are many levels of meaning in every sentence, requiring multi-level analysis techniques, or a **layered approach**, to understanding natural language text. The **morphological level** is concerned with individual word forms, removing suffixes and prefixes to produce word stems. The **lexical**

[1] Hugue, Casey, and Wood are affiliated with Trident Systems, Inc. , and Edwards is with NSWC.

level refers to procedures applied to full words, where a target word is compared to a dictionary or lexicon of known words. Lexical analysis is typically used in information retrieval to identify common words, such as "a" or "the", which have no information content with respect to the subject area addressed by the text. The preliminary identification of a word as a noun, verb, adjective, adverb, or other part of speech based on information in the dictionary is also a lexical operation, as it does not consider the words taken in combination. The **syntactic level** groups the words in a sentence into units, such as prepositional phrases, subject-verb-object, verb-adverb, and adjective-noun clauses. Sentence structure identification and analysis is positional, with no analysis of the meaning of the sentence. The **semantic level** uses word meanings and context information to restructure the text in a way that demonstrates the meaning of the sentence, such as actor-action verb-target. Finally, the **pragmatic level** uses real world, common sense, or domain specific knowledge to remove potential ambiguity in sentence analysis.

### Syntax and Semantic Analysis Systems

Several tools address syntax and semantic issues, but on a smaller scale than the information retrieval systems discussed in the next section. MetaLingual uses the SeCalc Translation System, comprising a modeling subsystem (with a dictionary, a dictionary manager, and an attributes manager), an English text translation subsystem, and a sentence-frame database manager and inferencing subsystem. These are used to generate entity-relationship diagrams and other information structures useful in assessing requirements. SeCalc employs sentence based analysis, using rules encoded in Clips Object Oriented Language (COOL), and supports word search, jargon definition, and the identification of objects and inter-object references.

AIRES, a requirements characterization and analysis tool under development at George Mason University, employs a sophisticated document parsing procedure which combines lexical and syntax analysis. All sentences and paragraphs in a document are assumed to be requirements; so, no capture procedure is implemented. Terms critical to extracting meaning from the natural language text are identified and weighted in the AIRES dictionary during the document conditioning phase. Paragraphs containing similar information are grouped together using syntax and semantic analyses applied within the context of the document terms identified as important. These groupings are determined by computing a numerical similarity measure between paragraphs of the requirements document based on the weights assigned to individual terms and predefined similarity thresholds. Ambiguous and incorrect or imprecise terms can be identified and replaced or corrected. Typically, the initial document conditioning is not sufficient to support the necessary semantic analysis; so, multiple reconditioning and similarity assessment phases are needed. Dictionary weights may need to be updated to ensure that paragraphs judged as semantically similar remain numerically similar. Functional and entity-relationship diagrams can be created at an impressive level of granularity. Since dictionary term weight assignment, document conditioning, and paragraph clustering must be performed manually, the current AIRES implementation may need to be extended to support requirements analysis of extremely large complex systems.

### Information Retrieval Systems

The purpose of information retrieval (IR) systems is to store large amounts of information economically and to provide efficient and timely extraction of requested information. While analysis is largely morphological and lexical, frequency information is used as an approximation of a word's importance to a document. Unique words in the document are identified, and the frequency of each such **term** in the document is calculated. These term frequencies are computed for the purpose of automatically extracting **index terms**, which are words in the document that encapsulate the key points of information in a document. Under this **auto-indexing** paradigm, extremely low frequency words and frequent but knowledge deficient words, such as "the", "and", "at", "or", " of" and "it", are deleted from the list to produce a set of index terms that will differentiate this document from other documents in a large set. This process is based on Zipf's law, which says that the term frequency times the rank of the term in decreasing frequency order should be constant. That is, the (uninterpreted) terms best suited to capture the key message in a document or document set are those of medium frequency. Since successful auto-indexing is very sensitive to the thresholds chosen for high and low frequency terms, the user often has the opportunity to adjust the index term set manually.

Commercial tools for retrieving information relevant to user queries, formed from index terms or on keyword and string searches, have been very successful. Systems such as Dialog, Medlars, and Lexis support retrieval of stored natural language text for use in reference analysis, in medical diagnosis and treatment, and in identifying and verifying legal precedents, respectively. Dialog takes queries in the form of keywords and a specific document database as inputs and returns a list of documents which are relevant to those keywords. Medlars classifies documents based

on a controlled language of medical terms. Queries must be converted to this controlled language before they can be applied to the document database. Thus, users must be very familiar with the term classification scheme to achieve the desired results. A benefit of Medlars is that symptoms of an illness can be entered, and potential diagnoses can be retrieved. This differs greatly from the Dialog system, whose goal is to identify documents which match a given query. Unlike Dialog, which uses a document representative, such as the title, author, and a few keywords, Lexis and Medlars support the full text of the document. In Lexis, the query language is unrestricted; however, familiarity with legal terminology is essential in constructing effective queries. None of these tools permits the manipulation of natural language text or the semantic sentence analysis needed to support requirements capture and analysis. Furthermore, all three systems need a trained archivist to form queries that achieve acceptable levels of document recall and precision.

## 3 Theoretical Basis of RECAP

Since IR techniques are designed to provide quick and accurate access to distinct subsets of very large information databases based on constraints or rules, they were chosen as the basis for the internal RECAP document representation. The current implementation of RECAP employs a loose interpretation of the cluster-based IR system paradigm in which important terms in the document are assigned weights which separate requirements sentences or paragraphs into sets sharing common terms. As described below, the goal of clustering in IR systems is to make it easier and quicker to access documents relevant to a given query, especially in a large database where it is not possible to keep all document representatives in main memory at the same time. This IR cluster methodology supports scalable efficient storage and access of text in a manner which allows document generation, test generation, and requirements tracing to various design, implementation, and maintenance tools for the system's entire life cycle. Surprisingly, this paradigm can be used to identify requirements that address specific domain areas, and to measure similarity between pragmatic information and requirements as stated in the document set under analysis. Note that the similarity computation used in AIRES to identify related paragraphs is also based in part on this paradigm. After an overview of the cluster paradigm, we describe its implementation in RECAP.

### Cluster Based IR Systems

In cluster-based information retrieval systems, the chosen index terms are the basis for a multidimensional coordinate system, referred to as **term space**, in which each position in the vector of all index terms for the document set corresponds to the presence or absence of a specific word or index term. The value corresponding to a given term in a specific vector is the **term weight**, which is chosen to reflect the importance of that term to the document represented by that specific vector. The weight of a specific term in a document vector is usually some function of the occurrence frequency of the term in the document and the number of documents that contain that term. If we were to graph the document vector in term space, the weight of a specific term would determine the coordinate of the document vector on the axis corresponding to that term. This construction permits a number to be assigned to the relationship between two documents or a query and potentially relevant documents. The **similarity coefficient** associated with a document and a query is a value based on the cosine of the angle between the corresponding vectors in term space. The closer the cosine is to unity, the more similar the document is to the query, because the angle between their representative vectors is approaching zero. This replaces the subjective assessment of a document's relevance to a query with an objective measure. The document set can be clustered based on these similarity coefficients to keep interrelated documents in close proximity, thus increasing the probability that the user will be able to identify most of the documents addressing a specific issue.

### Clustering in RECAP

In RECAP, we are dealing with a set of documents, but the meanings of sentences and paragraphs need to be determined. So, neither the standard nor cluster-based IR paradigms apply directly because no term can be deleted automatically from consideration. In the context of the clustered IR paradigm, each sentence is treated as a document representative, and assigned an appropriate vector in term space, with each word treated as an index term of weight 1. Conceptually, the master vector of terms relevant to the document is a subset of a "working dictionary" or Document Dictionary, which initially contains all unique words in the document set. RECAP supports requirements capture and analysis at the level of granularity specified by the user, whether sentence, paragraph, subsection, or section. Sentences with similar content are still mapped close together in term space. However, the queries applied to this representation are rules based on domain specific information, pragmatic information, and word lists which are designed to classify and analyze the requirements present in a specific document. The full power of the clustered IR paradigm will be explored in future work, when non-uniform weights will be used to represent differing levels of importance associated with system characteristics,

such as availability, maintainability, performance, safety, observability, and reliability to support dependability analysis. The baseline RECAP functions presented in the next section explain the use of the cluster paradigm in further detail.

## 4 RECAP Functional Overview

RECAP need not assume that a requirements document already exists, but supports requirements extraction from working notes or other sources in which the requirements are embedded. For example, a system engineer may be presented with full or partial documents, requiring different levels of effort to generate a full set of requirements. The baseline RECAP operational scenario, which is the focus of this section, assumes that a complete System Segment Specification (SSS) exists for the full system. RECAP is employed to identify, organize, restructure, and analyze the requirements contained in the SSS. Other scenarios to be supported in a future version of RECAP will assume that the full SSS must be generated from a partial SSS, from requirements only, or from scratch. The RECAP requirements and specification elicitation feature must be used to acquire the information needed by the baseline scenario.

Regardless of the scenario, a human needs to read, understand, and interpret a large amount of information. The RECAP tool is designed to augment the user's abilities for information organization and recall, supporting both sequential and random access to text and taking advantage of user insights to resolve difficulties associated with semantic analysis based on incomplete information. The novelty of RECAP lies in its use of multiple methods for managing and extracting meaning from a large information base. Erroneous, misleading, and incorrect terminology can be identified by applying rule-based queries, and corrections can be suggested. Domain-specific information can be added to the document dictionary to provide pragmatic knowledge.

### Document Parsing and Representation

For simplicity, we assume that all documents are available on-line in ASCII form. First, the text is parsed to identify individual paragraphs, sentences and words. Initially, all terms in the document are treated as being equally important; no common or "stop" words are deleted, as their presence may be required during the sentence content analysis. All words are stored in the Document Dictionary for reference throughout the document processing cycle. Based on user selectable parameters, individual sentences, paragraphs, or sections are numbered, representing the smallest unit of extraction (granularity of content), referred to as a sentence in the remainder of this discussion. The sentences are then assigned unique identifiers and stored in the RECAP document database, similar to the set of clustered vectors described in Section 3.

### Requirements Identification and Capture

Unlike other tools, RECAP does not assume that all sentences in a given set are requirements. Instead, individual sentences need to be evaluated by the user for the correct format and meaning, with RECAP support. For example, requirements can often be identified by the use of the terms "will", "shall", or "must". Statements that place limits on or constrain system functions, characteristics, and application parameters are also requirements.

Once a sentence is recognized as a requirement, the values of attributes associated with that requirement must be selected. RECAP provides several levels of automated assistance. In manual mode, the user must identify all sentences that are requirements and select the proper attribute values for each requirement; RECAP does not supply suggestions nor does it pre-set attribute values. In interactive mode, RECAP will identify potential requirements, based on predefined rules for recognizing them, and request the user's approval to capture them and to set potential attribute values. In this mode, default attribute values previously defined by the user can be chosen. In automatic mode, RECAP can identify and capture many requirements and guess at their attribute values with a specified level of confidence. However, user analysis and check-off are encouraged in this mode. Manual, interactive and automatic modes can also be selected for the other RECAP functions discussed in the remainder of this section.

### Domain-Specific Requirements

RECAP supports the identification and capture of requirements which are associated with functional and non-functional domains. The functional category of domains encompasses both the services that the system and its constituents are required to perform, such as sensing, actuating, monitoring, or supplying power. A domain-specific knowledge base is maintained in RECAP to aid in identifying domains relevant to the requirements document and to supply the pragmatic knowledge associated with certain requirements which may not be stated explicitly in the document. For example, a shipboard system would be affected by environmental conditions associated with the sea, such as salt water damage, turbulence associated with wave motion, and power interruptions due to generator failure. Critical system portions would need to be waterproof as well as resilient to the rocking motion of the ship. A backup power supply to compensate for generator problems might also be needed. RECAP supports the

development of domain-specific knowledge repositories to augment the information in requirements documents and to identify missing requirements.

Similar information can be stored for the non-functional domains, which address system characteristics such as dependability, performance and response time. The dependability domain addresses the system capacity to maintain the required services, even in the presence of faults, and includes constraints and characteristics such as availability, reliability, safety, and security. Dependability domain knowledge includes information specific to the stated system characteristics, such as the need to include a time interval if the system reliability is specified in the SSS. Queries based on this domain information can be applied to the vectors in document space to identify functions present in the SSS. User construction of rules and queries based on domain-specific information is also supported. The ability to access sequentially all the sentences in the document that satisfy a set of rules or a domain definition can increase the contextual information available to the user while attempting to understand the facet of the system related to the domain or rule which generated the list of sentences.

### Completeness, Correctness, and Consistency

Requirement consistency, correctness and, completeness play a major role in ensuring that the completed system satisfies the intentions of the specifiers. RECAP assists the user in identifying requirements which fail to satisfy any of these conditions through its rule-based query formation and application feature, described previously in the context of domain-specific analysis. Queries can be constructed to identify potentially ambiguous terms, such as "fast", "some", and "several". Rules can be written to identify missing information, such as units associated with numerical data, or a discussion of availability without any notion of repair, MTTF, or MTBF. Other rules can be written to detect requirements containing conflicting information, or violating physical, chemical or common sense laws. While the development of a standardized rule repository represents another research effort, a large set of rules based on term phrases, pragmatic knowledge, domain information, and definitions of essential terminology is currently available in RECAP. The support of user-defined rule definition and application provides the needed extensibility to the current prototype tool.

## 5 Discussion

The RECAP tool, with its capacity to aid the user in developing, identifying and analyzing large numbers of requirements, will support the user (or users in a multi-user scenario) in developing a set of clear, concise and consistent requirements. It will also generate a structured requirements document when given appropriate requirements template. The use of a clustered information retrieval paradigm for storing large sets of documents speeds up access to related portions of the documents and supports localized analysis.

RECAP is not intended to replace the human in identifying requirements and analyzing them. Instead, both sequential access to the text, and access based on a specific subject or domain area are supported. The RECAP ability to identify potential ambiguities and conflicts in an SSS is continually being expanded through the addition of new rules, new domain information, and the ability to apply multiple rules using Boolean combinations. In the future, we will add parts-of-speech identification to our parsing function, expand the domain data repository, explore the development of a structured rule-base, and continue to develop our user-friendly interface. We will also explore the benefits of incorporating mature semantics analysis techniques and other emerging technology into RECAP to provide a robust tool for capturing and analyzing complex system requirements.

### Acknowledgements

Support of this effort under Navy Contracts N60921-92-C-0131 and N00014-94-C-0085 is gratefully acknowledged. The extension of the cluster-based IR system paradigm to address portions of a document was funded by the George Washington University in 1990. We also wish to thank Dr. James Palmer and Dr. Richard Evans of George Mason University for sharing their insights into requirements analysis, acquired through the development and application of AIRES.

### References

[1] James Allen, Natural Language Understanding, Benjamin/Cummings Publishing Co., Inc., Menlo Park, CA (1987).

[2] Gerald Salton and Michael J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill Book Company, New York (1983)

# TRACING PRODUCT AND PROCESS INFORMATION WHEN DEVELOPING COMPLEX SYSTEMS

Stephanie White
Northrop Grumman Corporation
Mail Stop B38-35
Bethpage, NY 11714

## ABSTRACT

Military systems are usually large, distributed, and complex and are normally developed by a number of contractors. Engineers developing these systems require traceability. System engineers need traceability to control change, control the process, and control risk. Maintenance engineers need traceability to better understand the product and process used to develop the system. All engineers need quick access to information, and since the information is vast and complex, engineers need information abstraction and visualization techniques that help promote understanding.

Unfortunately tracing is difficult with current technology, and minimally supported. It is expensive to capture information, tools cross disciplines and do not have the same lexicon, we cannot measure the benefits of traceability, and the cost is high. We do not know the level of granularity at which to trace and how to present much of the information that should be traced, for example behavior and nonfunctional requirements.

This paper documents discussion of these issues during a panel session on traceability, that the author chaired at CSESAW '93. It discusses the importance of traceability, specifies some techniques for improving the state of practice in this area, and identifies problems that require further research. The paper also discusses a new method for tracing system behavior[1].

## 1. DEFINITION OF TRACE

The Standard for Defense System Software Development, DoD-STD-2167A defines traceability as a demonstration of completeness, necessity and consistency. Specifically, it defines traceability as follows: "(1) the document in question contains or implements all applicable stipulations of the predecessor document, (2) a given term, acronym, or abbreviation means the same thing in the documents, (3) a given item or concept is referred to by the same name or description in the documents, (4) all material in the successor document has its basis in the predecessor document, that is, no untraceable material has been introduced, and (5) the two documents do not contradict one another."

Webster's Third International Dictionary relates "tracing" to product artifacts as does DoD-STD 2167A, and relates "tracing" to process . It defines trace as "to follow step by step," and "to outline or present the development process or history of."

## 2. REASONS FOR TRACING

There are numerous reasons to trace system definition, system development, and their inter-relationships, including:

- Assessment of Quality and Status
- Checking for Consistency and Completeness
- Change Impact Analysis
- Maintenance
- System Evolution
- Process Improvement

Assessment of product quality and project status requires checking that requirements are met, risks are being addressed, the design is implemented properly, tests are adequate, planned tests are performed, and that the project is on schedule and within budget.

Checking for completeness, necessity, and consistency require traces of system definition elements. For example:

- Mission requirements should be traced to operational scenarios, system, subsystem, software, and hardware requirements.
- Objects, relationships, and attributes in a conceptual model should be traced to corresponding elements in the design, implementation, and test cases.
- High level objects, relationships, and attributes in a conceptual or design model should be traced to those at a detailed level.

Change impact analysis and system evolution require tracing inter-dependent information in the system definition, together with supporting information, for example issues, effectivity measures, tradeoffs, decisions, rationale, and priorities.

Process improvement requires tracing the process and its effects on system definition and implementation.

[1] This research was performed under contract to Naval Surface Warfare Center, Dahlgren Division, White Oak Detachment, and was funded under the Engineering of Complex Systems Technology Program sponsored by the Office of Naval Research, Contract Number N60921-93-C-0051.

## 3. OVERVIEW OF ISSUES

In most cases, DoD customers aren't asking for traceability except for that covered in DoD-STD 2167A. This standard addresses traceability of documentation (requirements, design, implementation, test cases) and traceability of unit Software Development Folders to component Software Development Folders.

Engineers need greater capability for traceability. They must be notified of changes and should rigorously trace the impact of changes on cost, schedule, and feasibility, on system design and implementation, on tests that must be repeated, and on support software and hardware. These engineers need automated support to assess whether the system is under or over designed, specifications contradict each other, lower level decisions are consistent with higher level decisions, test cases cover requirements, detailed behavior and high level behavior are consistent, and non-functional requirements are met.

Current traceability aids are reasonably good for product functions, but are limited for tracing system behavior and product attributes (DoD Software Technology Strategy, '91). Behavior is difficult to trace using current methods, as behavior is specified by stimulus-response statements that cross functional lines. Current trace mechanisms link stimulus-response paths to a significantly large system subset or the entire system.

Current methods manage product attributes such as timing and reliability as they would manage weight, by allocating a portion to a component. However, timing and reliability are emergent properties and cross functional boundaries. These properties cannot be analyzed by examining each component in isolation. Timing and reliability vary by scenario, and an unexpected event in one component can affect another.

Current methods trace high level requirements for reliability, availability, security, and safety directly to lower level requirements. In many cases, high level requirements for these attributes should be traced to a high level design policy, and that policy should be traced to the lower level requirements. For example, a high level requirement for hardware reliability could be satisfied by a policy of shadowed pairs or one of majority voting.

Different engineering roles use information for different purposes. There may even be a conflict between roles. For example, reliability engineers are interested in availability and safety engineers do not want any failures. For this reason the stakeholders themselves must decide what should be traced. Because there is a vast amount of information, these engineers need automated support for capturing and linking information while performing their normal engineering functions. All information may not have to be collected and stored. Some information can be derived automatically by smart tools.

## 4. TRACING REQUIREMENTS

Requirements should be traced to subsystem requirements, design, implementation, test cases, and test results. This is not a simple task due to the size and complexity of requirements and the many to many relationship between requirements and design elements. The following issues need to be addressed.

- How should requirements be structured to support maximum traceability?
- One requirement may trace to several design aspects:
  - What is needed to verify that a requirement has been satisfied?
  - How should traceability support this verification process?
- How should we track the rationale behind customer and derived requirements?
- What methods should be employed to trace behavioral requirements and product attributes?

Structuring requirements for maximum traceability.

It is not clear that any one requirements method is superior with respect to traceability. Each method (e.g., stimulus-response, object-oriented, data flow) provides some unique and necessary views. Most current methods trace requirements at too coarse a level of granularity. Traceability of events and conditions is needed, as well as traceability of data and functions.

Specifying method semantics in Entity-Relationship-Attribute (ERA) format supports the capture of models in databases and thus provides excellent support for traceability.

Verification that the design meets requirements.

One requirement may relate to many parts of the design, or many requirements may relate to one part. When one requirement relates to many parts of the design, the requirement may be divisible on a case by case basis. However, this is not always feasible, as with the requirement that something bad will not happen. We cannot enumerate all the bad things that could happen. In such cases, we should trace the requirement to a policy (e.g., for dealing with "bad things"), and then to design decisions that meet that policy.

Since we verify that requirements have been met using simulations, prototypes, tests, and analyses, we must trace requirements to simulations and studies that verify the design meets the requirements, as well as to tests that the implementation meets the requirements.

Rationale behind customer and derived requirements.

Customer requirements relate to the concept the customer has for meeting mission requirements, frequently captured in the operational concept

document or high-level system specification. Derived requirements result from contractor requirements analysis, and are derived due to incomplete customer requirements, or contractor decisions regarding high level policies. We should track the rationale behind both customer requirements and derived requirements, in case the mission changes, or feasibility, cost, and schedule risk become too high, driving a need to revisit requirements.

Tracing behavioral requirements.

Behavioral requirements are difficult to trace using current methods because current trace mechanisms link functions rather than threads. Methods that link functions associate stimulus-response requirements with significantly large system subsets or the entire system. Contractors should trace behavioral threads to more detailed threads (see White 94), and associate this behavior with test cases that drive simulations, prototypes, and implementations.

Tracing product attributes.

Product attributes are difficult to trace due to their pervasive nature.

Timing has numeric constraints, and is associated with normal and exceptional behavior paths that link input to output. Timing is also dependent on resource utilization so degraded conditions must be considered, vastly increasing the number of links.

Safe, survivable, and secure are rated by criticality level. Designers try to partition the system so that "highly critical" applies to a small part of the system, but this is difficult as we do not know how to create impenetrable boundaries. Verifying that attributes are met is equivalent to proving that bad things cannot happen, which is difficult or impossible. Therefore, strict design and coding principles must be followed, and we must trace flowdown and verification so that verifications can be repeated if a change occurs.

Fault tolerance is related to a pervasive philosophy (e.g., fail-operational/fail-safe, hot standby, compartmentalization), and to hardware and software design decisions. Tracing whether the fail-operational/fail-safe philosophy has been met is equivalent to inspecting all design and implementation. It is difficult to test a system for fault tolerance, as we cannot introduce every kind of fault. Every test that results in a failure should be analyzed for additional fault tolerance requirements, causing iterative test, requirements, design, code, re-test cycles, and iterative traces.

The pervasive nature of most product attributes indicates that it is not feasible to trace these attributes to the lowest level. To understand what must be traced, we need a defined process for specifying and verifying product attributes.

## 5. TRACEABILITY AND INFORMATION PARTITIONING

Today each discipline (e.g., avionics, software, communications) has its own language, methods, and tools. Lack of traceability results in errors. The following issues need to be addressed.

- Should the system database be partitioned according to discipline.
- Is partitioning feasible or is it preferable to have one system model?
- What is needed to trace information among different disciplines, tools, and databases?

Partitioning information according to discipline.

Today, it is not feasible to have one database containing all project information since each engineering discipline (e.g., avionics, software, communications) has its own language, methods, and tools. Currently, there is no single modeling method or language rich enough to represent all aspects of the system and still be understandable. For this reason we must understand the inter-relationships between views that affect tradeoffs, optimization, and interoperability.

Feasibility of partitioning.

Information that can be arranged in tree form is partitionable. Information that must be linked in network form is much more difficult to partition.

Tracing across disciplines and tools.

In order to trace information across different disciplines and toolsets, we need an ontology (a definition of the essential nature of the data) and a taxonomy of the information to be traced. This information is needed so that industry can standardize on the information that must be traced among tools and databases. The IEEE Computer Society Task Force on the Engineering of Computer-Based Systems is defining such an ontology. Tool developers are building tool wrappers and information ontologies that allow specific tools to exchange data.

## 6. PRIORITIZING WHAT SHOULD BE TRACED

The return on investment for traceability is not currently measurable, and establishing linkages for traceability can be labor intensive if these linkages are not established by tools during the development process. The following issues need to be addressed.

- How do we prioritize on elements to be traced?
- Is traceability dependent on the method used?

Studies needed.

We need studies of how people work in order to prioritize what they need traced. These studies must not interfere with project operations. A possible strategy is to examine artifacts such as change control

documentation and problem reports to determine where people spend time and what information is needed.

Limitations imposed by methods and tools.

Methods and tools help engineers structure their processes, but they also limit what is done. If a method does not include an activity, that activity will probably not be performed. If a tool does not capture information, that information probably will not be traced. Thus, the methods and tools that corporations choose both define and limit activities and information capture.

## 7. TOOL SUPPORT FOR TRACEABILITY

A number of tools support traceability. Some of these tools link information expressed by a single discipline within a single phase. Cadre Teamwork for Real-Time Structured Analysis is an example of such a tool. Other tools link information from multiple disciplines and phases. Cimflex's Product Track, Marconi's Requirements Traceability Manager (RTM), and Doors are examples of such tools. These tools usually use an ERA-like schema to capture information as the schema is easy to understand and readily extensible. The following issues need to be addressed.

- What are the issues in linking information from multiple disciplines and phases?
- What are the issues in capturing decisions and rationale?
- What additional capabilities are needed?

Issues in capturing information from multiple sources.

Information linked by tracing tools need not be specific to a model or discipline. For example, the engineer may want to link the estimated footprint, weight and power usage of a piece of computer equipment (stored in a hardware modeling tool) to the estimated throughput and memory requirements for a piece of software (stored in a software modeling tool). To efficiently use tracing tools, we should automatically transfer the information captured in tools that engineers use to perform their project specific functions (e.g., design, test, mange). Developing such interfaces is not easy. An ontological model is needed that formally expresses the underlying elements that are being linked and their relationships.

Many corporations are using Requirements Driven Design (RDD), (Ascent Logic '92), to document system conceptual models and Structured Analysis-Real Time (SA-RT), (Mellor '85, Hatley '87), to document software conceptual models. A partial ontology for linking information in RDD to information in SA-RT is documented in (White '93) This report identifies and compares RDD and SA-RT entities and relationships.

Issues in capturing decisions and rationale.

Some tools (e.g., RDD) support the capture of decisions and rationale as text, but many projects are not advocating the capture of this information as the cost/benefit ratio of doing this has not been proven. When rationale is captured, the information is usually sparse or partial, and difficult to use for further analysis by anyone other than the original writer. Further research is needed in this area.

Additional capabilities needed.

As indicated by Ramesh and Edwards (Ramesh '93), a model that addresses traceability issues should not only discuss linkages, but also the reasoning that can be performed with the linkage information, and approaches should capture the semantics of relationships (e.g., how the requirement is satisfied by the design). Tools do not have this capability today.

## 8. TRACING COMPLEX SYSTEM BEHAVIOR

Existing methods that organize system behavior by function and object provide poor visibility of inter-functional threads through the system. Because behavior is divided among a number of functions, system engineers find it difficult to verify that the entire thread from environment stimulus to system response is complete, that inter-component aspects are consistent, and that no unnecessary functions are included. This inability to verify the completeness and consistency of inter-functional threads is inherent in existing methods, and is a problem even when one engineer is creating the design. The problem is compounded on a large system with several contractors and numerous designers.

Engineers need a method for identifying critical stimulus-response threads that cut horizontally across objects and functions in the requirements model. They need a method that can trace these system threads to more detailed threads, and to threads in the design and implementation.

Our research solves problems in tracing system behavior. We have defined a new method, Concurrent Stimulus-Response Threads (CSRT), for tracing threads (White '93). CSRT represents system behavior in a manner that can be traced throughout the system. These threads of behavior relate system stimulus to system response, and relate environment and system events with system functions.

## 9. CSRT

Our approach, Concurrent Stimulus-Response Threads (CSRT) creates a firm foundation for tracing and analyzing critical threads. The highest level threads are the system requirements. These threads specify the

environment stimulus and the system response. More detailed requirement and design threads define paths of system operation from system stimulus to system response that implement the high level requirement. Once the system stimulus occurs, other environment or system events can occur (e.g., a failure). Each thread is uniquely determined by a specific sequence of these conditions and events. The threads specify the behavior of the system in response to the conditions and events.

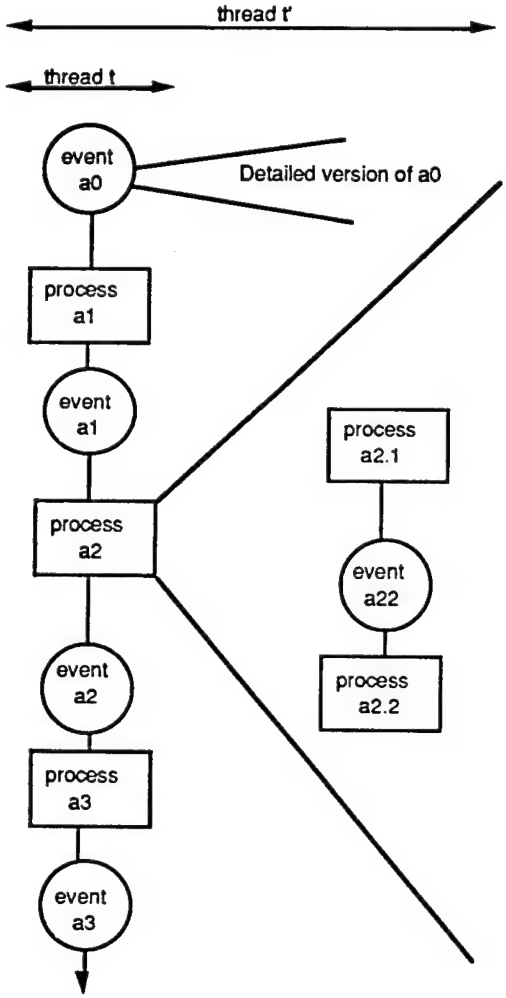



Figure 1 Tracing Threads

We define a "simple thread" of behavior T as an alternating sequence of events and processes,

$$T = E_1, P_1, E_2, P_2, E_3, P_3, ..., E_n, P_n$$

where an event $E_i$ is a compound expression of events and conditions or a sequence of events.

Examples:

$E_i = E_1$ when $C_1$ OR $C_2$

$E_i = E_1$ when $C_1$ AND $C_2$

$$E_i = E_1, E_2, E_3 ... E_n$$

In a "simple thread", a process $P_j$ is a sequence of processes or a set of concurrent processes, but cannot consist of alternate processes as disjuncture indicates separate simple threads or a compound thread.

Examples:

$P_j = P_{j1}, P_{j2}, P_{j3} ... P_{jn}$

$P_j = P_{j1}$ & $P_{j2}$ & $P_{j3}$

In a simple thread, Pj cannot be represented as $P_{j1}$ OR $P_{j2}$.

Assume a thread T:

$$T = E_1, P_1, E_2, P_2, E_3, P_3, ..., E_n, P_n$$

A more detailed version of T, called T' will include the events and processes of T. In T', the events may be expressed in terms of more detailed data items, the processes may be expanded into more detailed threads, and new events and processes may be added (e.g., to handle interfaces due to allocation), see Fig. 1. Each thread T should be traced to the more detailed thread T'.

Our concept is that threads should be generated from existing models, for example from RDD, RT-SA, or object oriented methods. We do not recommend developing a new modeling method based on threads. In Fig. 2, we show several threads for a soda vending machine that were manually generated from an RT-SA model by examining data flow diagrams and unraveling state machines. In (White '93) we also generated threads from an RDD model of the soda vending machine. The methods used for generating threads from RT-SA and RDD models could be automated. Figure 2 shows threads of soda vending machine behavior, at several levels of abstraction. At the top level, we have two concurrent threads, Thread 1: React to Coin, and Thread 2: React to Customer Selection. Thread 2 traces to two threads, Thread 2.1: React to Coin Return Request, and Thread 2.2: React to Product Selection. Threads 2.1 and 2.2 are alternate threads for the entire Thread 2. They are not an expansion of a particular function in Thread 2. Thread 2.2 traces to two alternate threads, Thread 2.2.1: React to Product Selection and Payment Invalid, and Thread 2.2.2: React to Product Selection and Payment Valid. Note that each thread can be uniquely identified by a specific sequence of conditions and events.

In this paper we trace threads within the same model, but related threads should also be traced between requirements, design, implementation, and tests. Developers need these thread views of system behavior to isolate and understand different scenarios of system operation, and to be sure that threads are consistent and requirements are met.

| Thread 1 (REACT TO COIN) | Thread 2 (REACT TO CUSTOMER SELECTION) |
|---|---|
| Event 1: Coin<br>Process 1: Accept & Count Coins | Event 1: Customer Selection<br>Process 1: Respond to Customer Selection |

| Thread 2.1 (REACT TO COIN RETURN REQUEST) | Thread 2.2 (REACT TO PRODUCT SELECTION) |
|---|---|
| Event 1: Customer Selection is Coin Return<br>Process 1: Respond to Return Coin Request | Event 1: Customer Selection is Product Selection<br>Process 1: Provide Product |

| Thread 2.2.1 (REACT TO PRODUCT SELECTION & PAYMENT INVALID) | Thread 2.2.2 (REACT TO PRODUCT SELECTION & PAYMENT VALID) |
|---|---|
| Event 1: Customer Selection is Product Selection<br>Process 1: Validate Payment<br>Event 2: Payment Invalid<br>Process 2: Return Coins | Event 1: Customer Selection is Product Selection<br>Process 1: Validate Payment<br>Event 2: Payment Valid<br>Process 2: Dispense Product & Change |

Figure 2 Soda Vending Machine Threads

## 10. FUTURE WORK

We intend to integrate formal methods for specifying thread interdependencies with CSRT, and develop methods for analyzing thread consistency. In Fig. 1, T represents a single thread, but complex systems are composed of multiple concurrent threads with interdependencies. These threads must either include synchronization points, may be restricted by "constrained expressions" that specify the required ordering of events (Wiledon '86), or may be restricted by using a combination of conditions and events as in the Naval Research Laboratory Software Cost Reduction (SCR) Method (Heninger '80). We will investigate these techniques for use with CSRT. We will also investigate the use of temporal logic to analyze whether the sequence of events and functions is consistent in related threads.

## 11. CONCLUSIONS

Good process and product traceability would reduce costs significantly. Contractors should place more emphasis on the need to improve information capture and traceability. Tracing critical stimulus-response requirements to threads in requirements models, design and implementation would locate inter-functional and inter-component errors, inconsistencies, and missing elements. We plan continued experimentation with CSRT on larger problems.

## 12. ACKNOWLEDGMENTS

The author would like to thank Mike Edwards, Naval Surface Warfare Center, for his critique of this research and helpful suggestions.

## 13. REFERENCES

Ascent Logic Corp., *RDD Manual*, Jun. 1992.

Hatley, D.J. and Pirbhai, I.A., *Strategies for Real-Time System Specification*, Dorset House, NY, 1987.

Heninger, K., "Specifying Software Requirements for Complex Systems: New Techniques and their Application", *IEEE Trans. on Software Engineering*, Vol. SE-6, Jan. 1980, pp. 2-13.

Mellor, S.J. and Ward, P.T., *Structured Development for Real-Time Systems*, Vol. 1-3, Prentice Hall, Englewood Cliffs, NJ, 1985.

Ramesh, B. and Edwards, M., "Issues in the Development of a Requirements Traceability Model, IEEE International Symposium on Requirements Engineering, IEEE CS Press, Jan. 1993.

White, S.M., *A Pragmatic Method for Computer System Definition*, PhD Dissertation, Polytechnic University, University Microfilms, Ann Arbor, MI, Jun. 1987; also Tech. Report RE-791, Grumman Corporate Research Center.

White, S.M., "Tracing Entities, Relation-ships, and System Behavior", Final Report, NSWC Contract No. N60921-93-C-0051, Grumman Corporate Research Center, October, 1993.

Wiledon, J.C., "Applying Event Based Analysis to Specification and Designs", IFIP 1986, Elsevier Science Publishers, pp. 577-581.

# System Dependability Assessment Tool

Eric W. Brehm
Advanced System Technologies, Inc.
12200 E. Briarwood Avenue, Suite 260
Englewood, CO 80112
(303) 790-4242

## Abstract

Dependability is an increasingly important aspect of the Navy's mission critical computer systems. These systems must be maintained in a state of readiness to support critical mission functions, and must be able to perform these functions correctly, despite the fact that system components may fail to operate as intended during operation of the system. Automated tools are needed to assist in specifying and evaluating dependability characteristics of Navy computer system designs, and in balancing dependability against other system attributes such as performance, security, and cost.

The System Dependability Assessment Tool (SDAT) is an automated tool for analysis of mission critical computer system dependability characteristics. SDAT consists of a dependability specification component, that allows interactive creation and manipulation of system design representations, and a dependability evaluation component, that automatically translates system design descriptions into mathematical models, and computes a range of quantitative dependability metrics.

A preliminary version of SDAT has been developed that encompasses traditional reliability, maintainability, and availability (RMA) analysis. Future work will extend these capabilities to address other aspects of dependability assessment. SDAT is being developed with the ultimate goal of full integration with other Navy system design capture and optimization tools.

## 1. Introduction

System *dependability* is an essential characteristic of mission critical computer systems, that refers to the degree of confidence that can be placed in the services provided by a system. Dependability relates both to the operational readiness of a system to support mission functionality, and to the ability of the system to perform these functions correctly despite imperfections (faults) in system components, and in the environments in which the systems operate. Since unreliable system behavior can threaten mission success, and may in some situations even have catastrophic consequences, increasingly stringent dependability requirements are being placed on the Navy's computer systems.

With the widespread use of parallel and distributed processing architectures for Navy computer systems currently under development, faults will be increasingly likely to occur during system operation. At the same time, the increasing complexity of these systems will make it more difficult to determine the effects of component faults on system functionality, and to design fault detection and recovery mechanisms and redundancy management schemes that will ensure sufficient levels of overall system dependability.

Due to these inherent complexities, automated tools are needed to help predict the behavior of a system from a dependability perspective, and to tailor design features as necessary to ensure that all dependability requirements will be met by the system when operational. Dependability assessment activities supported by these tools include dependability *specification*, which involves describing the inherent behavior and proposed design of a system from a dependability point of view, as well as

dependability *evaluation*, which involves the application of mathematical modeling techniques to produce quantitative estimates of system dependability.

This paper describes the System Dependability Assessment Tool (SDAT), an automated tool that addresses dependability specification and evaluation of mission critical computer system designs. The overall goal of the SDAT development effort is to provide automated capabilities that will help system engineers gain insights into the dependability implications of alternative design decisions, and thus enhance the Navy's ability to deliver mission critical computer systems that meet stringent dependability requirements.

Section 2 describes general characteristics of the SDAT tool that are intended to enhance its usefulness for dependability assessment of Navy systems. Sections 3 and 4 summarize the specific dependability specification and evaluation capabilities that have been incorporated in SDAT to date, and Section 5 outlines plans for future research and development.

## 2. SDAT Tool Overview

SDAT separates the concerns of dependability specification and dependability evaluation by providing separate, and to some extent independent, capabilities in each of these areas. In this *multi-level design assessment* approach, illustrated in Figure 1, dependability specification is based on system representations expressed at a level of abstraction familiar to system designers, while dependability evaluation is based on mathematically oriented system representations. (For example, whereas designers tend to think in terms of such elements as "components" and "functions," mathematical models are typically expressed in terms of system "states" and "transitions" between these states.)

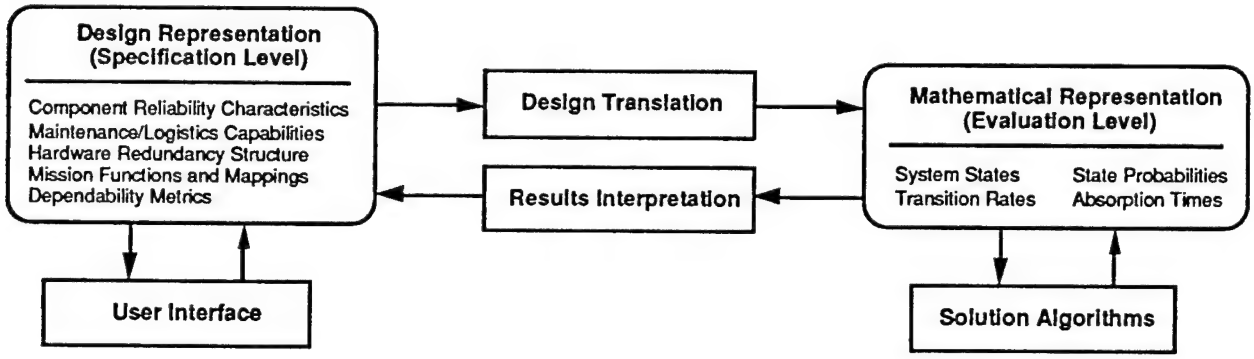



*Figure 1: Multi-Level Design Assessment in SDAT*

The interface between SDAT dependability specification and dependability evaluation capabilities is managed by facilities that automatically *translate* design representations into equivalent mathematical models, and by facilities that *interpret* the computational results that are obtained after application of one or more model solution algorithms. Ultimately, dependability evaluation results are presented as system dependability *metrics* that can be expressed in the dependability specification domain. (For example, "steady-state probabilities" computed for a Markov chain model of a system may be transformed into an "availability" dependability metric that is associated with a particular mission capability supported by the computer system.) The process of attaching dependability evaluation results to elements of a design description is often referred to as "back annotation."

From a user's point of view, the multi-level tool structure adopted for SDAT simplifies the process of dependability assessment by eliminating the need to specify mathematical elements of models directly. Comparative evaluation of alternative designs is easily accomplished through manipulation of high-level design attributes, with the corresponding variations in model representation handled automatically through translation. In addition, a single design description can be translated into multiple model

representations, so that alternative solution techniques may be selected as appropriate for evaluation of a particular design.

The SDAT tool architecture addresses the need to accommodate future enhancements and extensions to the tool's components. In particular, new types of data elements can be added to design representations with virtually no modification to the software responsible for retrieval, manipulation, display, or storage of design information. Also, the dependability evaluation capability is structured in such a way that external evaluation facilities—such as commercially available mathematical solution packages—can be easily accommodated within the computational apparatus of the tool.

## 3. SDAT Dependability Specification Component

From a dependability perspective, a computer system may be viewed as a collection of components, organized into subsystems that provide the functions or services for which the system is designed. A mission critical computer system must provide these functions or services with a very high degree of confidence, despite the fact that the system's constituent components may fail either prior to or during a mission. Thus, examination of the following basic issues is an essential starting point for dependability assessment:

- What is the *inherent reliability* of the system's hardware components, i.e. what is the likelihood that a given component will fail during the time that it is needed to support critical system functionality?
- What *maintenance* capabilities, including spare parts inventories, will be provided to restore failed hardware components and configurations, via repair and/or replacement actions, to an operational state?
- How should *structural redundancy* be provided in the system architecture, i.e. how should the system's hardware components be organized into logical and/or physical configurations, such that functional continuity can be maintained to an acceptable degree after a component failure occurs?

These questions are the basis for what may be called system reliability, maintainability, and availability (RMA) analysis, where *reliability* relates to the ability of the system to operate correctly for a specified period of time (e.g., in support of a particular mission), *maintainability* relates to the ability of the system to restore components and functional capabilities that have failed, and *availability* relates to the likelihood that the system is an operational state at a given moment of time (e.g. at the beginning of, or at some critical point during, a mission). These three basic aspects of dependability are closely related; for example, increasing either reliability or maintainability tends to improve availability. Depending upon the system or aspect of a system under consideration, any or all of the basic RMA "ilities" may be of particular importance.

The initial dependability specification capability developed for SDAT is oriented toward capturing system design characteristics that are relevant to system RMA analysis. The design representation schema that is used to capture this information is expressed as a dependability design *meta-model.* (Generally, a meta-model is a schema that describes the structure and types of information contained in a class of models [1]). The SDAT meta-model is based on an entity-relationship-attribute-plus-inheritance framework that is consistent with the Case Data Interchange Format (CDIF) standard currently being formalized by the Electronic Industries Association (EIA) [2]. Compliance with the EIA CDIF standard offers the potential for SDAT, ultimately, to be able to exchange system design information with other system engineering tools that comply with the same standard.

The SDAT dependability design meta-model supports specification at three levels: component, configuration, and mission. The design information that is specified in each of these areas is summarized briefly below.

Components. At the component level, the basic unit of specification is called a *line replaceable unit (LRU)*; these are the physical devices that are repaired or replaced when a failure occurs during system operation. The rate at which components fail during operation is specified in terms of a *Mean Time to Failure* parameter for each LRU type. Component maintainability characteristics are specified in terms of a

*Mean Time to Repair* parameter, or the amount of time required to restore a failed unit to an operational state, a *Mean Time to Replace* parameter, or the amount of time required to remove and replace a failed on-line unit, and a *Sparing Level*, or the number of additional units of a given type initially available to replace failed units.

Dependability metrics that are computed by SDAT for LRU types, and for which separate "slots" are provided in the dependability design meta-model, are *Availability*, or the probability at a random instant of time that an on-line unit of a given type is operating correctly, and *Mean Time to Restore*, or the average time required to restore operational capability to a failed unit, taking into account the fact that an on-line component which fails when spare parts inventories are exhausted may have to wait for physical repair actions to be completed before functional capability is restored.

Configurations. The organization of components into configurations to provide structural redundancy is specified in SDAT using *reliability graphs*. These diagrams (which can be expressed equivalently as "fault trees") indicate the ability of the system to reallocate functionality to alternative components after failures occur. To account for the fact that switchover of functionality may occur at multiple levels of aggregation, and to simplify specification and analysis of the graphs themselves, SDAT allows reliability graphs to be defined hierarchically, i.e. a reliability graph may contain other graphs. Reliability graphs may be either of series-parallel (decomposable) form (including "K-of-N" type specifications), or may be arbitrarily complex. Dependability metrics computed for reliability graphs include *Availability*, *Mean Time to Failure*, and *Mean Time to Restore*, which correspond to, respectively, the basic RMA metrics steady-state availability (or the long-run fraction of time that the configuration is operational), average uptime, and average downtime.

Missions. SDAT also uses reliability graphs to represent the *mission capabilities* that are to be supported by the computer system. The *Mission Time* parameter specified for each mission capability represents the continuous interval of time during which correct operation of the system is required to ensure successful accomplishment of that aspect or phase of the mission. The *Mission Reliability* parameter holds back annotated mission-based (transient) dependability results; it represents the probability that no critical failure occurs in the computer system during the specified Mission Time.

## 4. SDAT Dependability Evaluation Component

The dependability design information discussed in the previous section, including line replaceable unit (LRU) reliability and maintainability characteristics, hardware redundancy structures, and critical mission capabilities, provides the basis for quantitative dependability models that compute estimates of system reliability, maintainability, and availability. In the current version of SDAT, two different types of analysis are performed: steady-state and transient.

*Steady-state dependability analysis* is concerned with the overall readiness of a computer system to support its operational requirements. Given the cycle of failures, repairs, and on-line replacements taking place at the LRU level, and the cycle of failures and reconfigurations (switchovers) taking place at the configuration level, the primary goal is to determine the percentage of time that sufficient hardware is operational to support critical mission functionality; steady-state metrics applicable at the LRU and configuration levels may also be of interest.

The steady-state dependability metrics that are calculated for each LRU type by SDAT, as mentioned in the previous section, are Availability and Mean Time to Restore (MTTR). For each configuration and mission capability, the metrics computed are Availability, Mean Time To Failure (MTTF), and Mean Time to Restore (MTTR). (For LRUs, Mean Time to Failure is a measure of inherent reliability, or mean component lifetime, and is specified by the user as part of a dependability design description.) For LRUs, configurations, and mission capabilities, the terminology used above is consistent with the "standard" definitions in use, i.e.:

$$\text{Availability} = \frac{\text{mean uptime}}{\text{mean uptime} + \text{mean downtime}} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

The approach used in SDAT for evaluation of steady-state dependability metrics, illustrated in Figure 2, is to first perform analysis at the LRU level, then use the component-level results to obtain results for the reliability graphs corresponding to configuration types and mission capabilities.

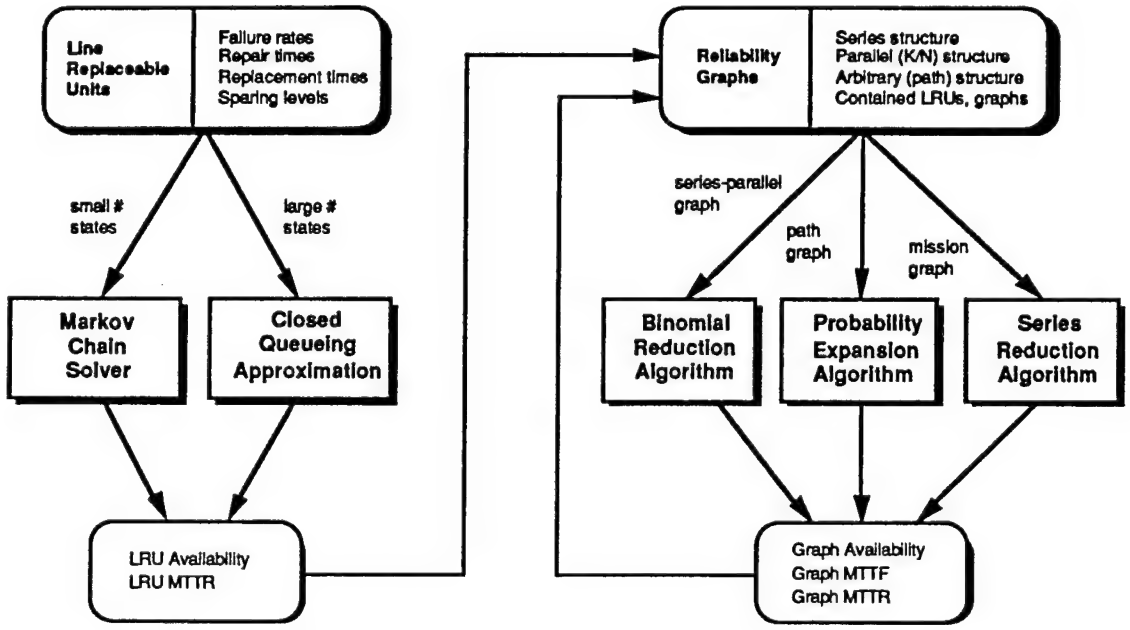



*Figure 2: Steady-State Dependability Analysis*

LRU Availability and MTTR results are computed as a function of LRU failure rates, repair times, replacement times, and sparing levels, using a Markov chain based LRU RMA model. Either one of two different computational methods (an exact Markov chain solver, and a approximate solution method based on state aggregation techniques that has been used to analyze closed queueing networks [3]) may be used for a given class of LRUs, depending upon the size of the underlying mathematical model.

Steady-state dependability results for each reliability graph are computed as a function of the graph's topological structure, as well as the results computed for the graph's constituent elements (LRUs or other reliability graphs). For series-parallel configurations, a binomial reduction algorithm is applied [4], and for arbitrarily complex configurations (source-terminal path structures), a union of tie sets probability expansion algorithm [5] is used. A simple series reduction algorithm is used to compute steady-state results for each mission capability graph.

*Transient dependability analysis* is concerned with dependability behavior occurring within a fixed interval of time, such as during a critical phase of a mission, or during an entire mission. For each mission capability defined, based on the specified mission duration and on the particular configurations required for mission success, SDAT calculates Mission Reliability, or the probability that sufficient operational hardware remains available to support the mission capability throughout its duration. It is assumed that switchover to backup components can occur during the mission (as expressed in the reliability graph corresponding to the mission capability), but that physical repair and or replacement of LRUs does not take place during the mission. This means that failure of an LRU that is required for the mission, but for which there is no on-line backup, will cause immediate failure of the mission.

Upon completion of steady-state calculations for each LRU, configuration type, and mission capability, and transient calculations for each mission capability, the dependability results produced by SDAT are back annotated in the dependability design specification, by instantiating the appropriate parameters with the values determined during the evaluation.

## 5. Future Work

Future work on SDAT will address three primary technical areas: (1) dependability specification enhancements, (2) dependability evaluation enhancements, and (3) integration of SDAT with other Navy information models and system engineering tools.

In the area of dependability specification, among the areas where extensions to the SDAT dependability design meta-model are needed are: (a) more detailed specification of mission phases and operational modes, including the effects of dynamic task reallocation and/or functional reconfiguration schemes, (b) additional maintenance and logistics parameters, including multi-stage repair facilities and scheduled maintenance policies, (c) representation of transitions between operational states related to fault detection, structural reconfiguration, and functional recovery capabilities, (d) representation of the effects of transient hardware faults, software faults, and human reliability factors.

In the area of dependability evaluation, extensions that are needed to the suite of methods currently in SDAT include the following: (a) methods to analyze multiple mission phases and/or operational modes, (b) expanded analytic or simulation based methods for analyzing complex system maintenance and logistics specifications, (c) methods to analyze the effects of fault handling behavior and other "fast transition" states, (d) approximate methods for bounding dependability results in complex reliability graphs, and (e) methods to compute "worst case" dependability results, such as percentiles of down time based on failure and repair probability distribution information.

A continuing objective of the SDAT development effort is to provide for interoperability between SDAT and other Navy models, methods, and system engineering tools. Proposed steps to accomplish SDAT integration in the future include: (a) integrating the SDAT dependability design meta-model with system design capture information models currently being developed within the Engineering of Complex Systems (ECS) block program, (b) adding a CDIF import/export component to SDAT to provide for file exchange with other tools supporting the CDIF standard, and (c) developing an application program interface (API) that will allow SDAT to be embedded in other tools, for example as a computational engine that can be accessed by design optimization tools.

## Acknowledgements

The development of the SDAT tool described in this paper was sponsored by the Naval Surface Warfare Center - White Oak Laboratory, Silver Spring, Maryland.

## References

1. M. Blaha, "Models of models," *Journal of Object Oriented Programming*, September 1992, pp. 13-18.

2. Electronic Industries Association CDIF Technical Committee, "Framework for Modeling and Extensibility," EIA-PN2387, May 1991.

3. C. H. Sauer and K. M. Chandy, "Approximate solution of queueing models," *IEEE Computer*, Vol. 13, No. 4, April 1980, pp. 25-32.

4. P. A. Kullstam, "Availability, MTBF, and MTTR for a repairable M out of N system," *IEEE Transactions on Reliability*, Vol. R-30, No. 4, October 1981, pp. 393-394.

5. W. G. Schneeweiss, "Computing failure frequency, MTBF, and MTTR via mixed products of availability and unavailabilities," *IEEE Transactions on Reliability*, Vol. R-30, No. 4, October 1981, pp. 362-363.

# Synthesis of Multilevel Security and Timing Requirements in Complex Real-Time Database Systems†

Sang H. Son and Rasikan David
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
(Ph) 804-982-2205 (Fax) 804-982-2214
son@virginia.edu

**ABSTRACT**

Database systems for real-time applications must satisfy timing constraints associated with transactions, in addition to maintaining data consistency. In addition to real-time requirements, security is usually required in many DoD-related systems, because sensitive information must be safeguarded. In general, the requirement of multilevel security conflicts with the timing requirements of the database. Decisions in transaction scheduling and conflict resolution for concurrency control in real-time databases do not consider security levels that are critical to support multilevel security. This paper addresses transaction scheduling and concurrency control in real-time secure databases - one of the critical issues that must be investigated to support multilevel secure database systems for real-time applications.

## 1. Introduction

A real-time database management system (RTDBMS) is a transaction processing system where transactions have explicit timing constraints. Typically, a timing constraint is expressed in the form of a *deadline,* a certain time in the future by which a transaction needs to be completed. A deadline is said to be *hard* if it cannot be missed or else the result is useless. If a deadline can be missed, it is a *soft* deadline. With soft deadlines, the usefulness of a result may decrease after the deadline is missed. In RTDBMS, the correctness of transaction processing depends not only on maintaining consistency constraints and producing correct results, but also on the time at which a transaction is completed. Transactions must be scheduled and processed in such a way that they can be completed before their corresponding deadlines expire. Real-time database systems are being used for a variety of applications such as process control, mission critical applications in command and control systems and radar systems, computer integrated manufacturing systems, and air traffic control systems, among others.

Conventional data models and databases are not adequate for time-critical applications, since they are not designed to provide features required to support real-time transactions. They are designed to provide good average performance, while possibly yielding unacceptable worst-case response times. Very few of them allow users to specify or ensure timing constraints. During the last few years, several research and development efforts have been reported on RTDBMSs [SIGM88, HUAN91, KIM93, SHA91, SON90, SON91, SON92, SON92b, SON93, SON94].

While the advances were being made in the area of RTDBMS during the past decade, much work has been carried out on the design and development of a multilevel secure database management system (MLS/DBMS), mainly based on the relational data model (see, for example, [AFSB83, DENN87, STAC90]). As a result, at present, some of these MLS/DBMSs are commercially available. In these database systems, users cleared at different levels are expected to access and share a database consisting

† This work was supported in part by ONR, by Loral, and by CIT

of data at different sensitivity levels. The MLS/DBMS must ensure that users obtain data classified at or below their security levels.

Most of these MLS/DBMSs are expected to be used for $C^3I$ applications. In principle, however, any database system that maintains sensitive information to be shared by multiple users with different levels of security clearance requires multilevel security. Many of these applications also require RTDBMSs. As more and more of such systems are in use, one cannot avoid the need for integrating them. That is, the RTDBMSs have to be made multilevel secure and the MLS/DBMSs need to support real-time applications. Not much work has been reported on developing DBMSs which supports both the synthesized requirements of multilevel security and real-time. Recently, at the first IEEE Workshop on Real-Time Applications, we presented a position paper on issues for supporting both requirements [SON93b]. Our focus was on the security impact on real-time transaction processing as well as discuss architectural issues and data modeling issues to support real-time applications which require multilevel security. In this position paper, we concentrate on the issue of transaction scheduling and outline our approach to achieving trade-offs between security and real-time requirements.

## 2. Transaction Scheduling for Multilevel Security

Concurrency control is used in databases to manage the concurrent execution of operations by different subjects on the same data object such that consistency is maintained. In multilevel secure databases, there is the additional problem of maintaining consistency without introducing covert channels. A concurrency control algorithm is said to be fair if the same level of service is provided to all classes of transactions, where the definition of service varies with the application. For a real-time system, it could signify the guaranteeing of a deadline, while in a conventional database system it could be the response time for each transaction.

Timing constraints have been found to have conflicts with consistency requirements. As a result, various transaction processing algorithms have been adapted to support real-time applications [HUAN91, SHA91, SON91, SON93]. Similarly, since security constraints have also been found to have conflicts with consistency requirements, various transaction processing algorithms have been adapted for a multilevel environment [THUR92].

Covert channel analysis and removal is the single most important issue in multilevel secure concurrency control. The notion of non-interference has been proposed as a simple and intuitively satisfying definition of what it means for a system to be secure. The property of non-interference states that the output as seen by a subject must be unaffected by the inputs of another subject at a higher access class. This means that a subject at a lower access class should not be able to distinguish between the outputs from the system in response to an input sequence including actions from a higher level subject and an input sequence in which all inputs at a higher access class have been removed.

We have developed a secure two-phase locking protocol (S2PL) to address the requirement of multilevel security in transaction scheduling and concurrency control. Instead of presenting the S2PL protocol, we briefly outline the modifications made to the 2PL protocol as follows. A detail description of the S2PL and its correctness proof can be found in [DAVI93].

Consider the two transactions, T1 (SECRET) and T2 (UNCLASSIFIED). T1 has set the read lock on $x$ and T2 needs to set the write lock on $x$. Basic two phase locking would fail to support security requirement because T2 would be blocked because of the readlock of T1. In S2PL, three different kinds of locks are used to resolve conflicts among transactions: real locks, virtual locks, and dependent virtual locks.

(1) Real Lock (of the form pli[x]): A real lock is set for an action pi[x] if no other conflicting action has a real lock or a virtual lock on x.

(2) Virtual Lock (of the form vpli[x]): A virtual lock vpli[x] is set for an action pi[x] if a transaction at a higher access class holds a conflicting lock on x (pi[x] has to be a write to satisfy the Bell-

LaPadula properties). The virtual lock is non-blocking. Once a virtual lock vpli[x] is set, pi[x] is added to queue[x] and he next action in Ti is ready for scheduling. When pi[x] gets to the front of the lock queue, its virtual lock is upgraded to a real lock and pi[x] is submitted to the scheduler.

(3) Dependent Virtual Lock (of the form dvpli[x]): A dependent virtual lock is set for an action pi[x] (where p is a write) if a previous write wi[y] in the same transaction holds a virtual lock. An action pi[x] which holds a dependent virtual lock with respect to another action wi[y] is not allowed to set a real lock or a virtual lock unless wi[y]'s virtual lock is upgraded to a real lock.

## 3. Approach towards Synthesis of Security and Real-Time Requirements

Preliminary results from the performance analysis of Secure 2PL exhibit a much better response time characteristic than Secure OCC. Its operating region (the portion of the curve before the saturation point) is much larger than that of Secure OCC. Further, staleness is not an issue in Secure 2PL as with Secure MVTO. However, this alone does not suffice when timing constraints are present on transactions. In Secure 2PL, transaction scheduling order is determined purely by the order in which transactions acquire locks. No conscious effort is made to schedule transactions according to their priority, or according to how close a transaction is to meeting its deadline. In a real-time database system this is unacceptable. It is our claim that the security properties have to be sacrificed to some extent to ensure a certain degree of deadline cognizance.

A covert timing channel is opened between two collaborating transactions - one at a higher access class and the other at a lower access class - if the higher access class transaction can influence the delay seen by a lower access class transaction. The bandwidth of a covert channel is a measure of how easy it is for the higher access class transaction to control the delay seen by the lower access class transaction. If there is a great degree of randomness in the system, i.e., an indeterminate number of transactions could be affecting the delay that the higher access class transactions wants a lower access class transaction to experience, then the bandwidth is low. On the other hand, if the higher access class transaction knows that the lower access class transaction to which it wants to transmit information is the only other transaction in the system, then the bandwidth is infinite. Therefore, when security has to be sacrificed, a policy that keeps the bandwidth of the resulting covert channel to a minimum is desirable. To ensure this, the security policy has to be adaptive, i.e., determining whether security is to be violated or not when a conflict arises should depend on the current state of the system and not on a static, predecided property. Our adaptive policy to resolve conflicts between lock holding and lock requesting transactions is based on past execution history. Whenever a transaction T1 requests a lock on a data item $x$ on which another transaction T2 holds a conflicting lock, there are two possible options:

- T1 could be blocked until T2 releases the lock.
- T2 could be aborted and the lock granted to T1.

If T1 were at a higher security level than T2, the latter option would be a violation of security. However, if T1 has greater priority than T2, then the latter option would be the option taken by a real-time concurrency control approach. In our approach, we strike a balance between these two conflicting options by looking up past history. A measure of the degree to which security has been violated in the past is calculated. A similar measure of the degree to which the real-time constraints have not been satisfied can be obtained from the number of deadlines missed in the past. These two measures are compared and depending on which value is greater, either the security properties are satisfied or the higher priority transaction is given the right to execute.

The two factors that are used to resolve a conflict are:

- Security Factor (SF): (number of conflicts for which security is maintained/ total number of conflicts) × difference in security level between the two conflicting transactions.

• Deadline Miss Factor (DMF): number of transactions that missed their deadline/total number of transactions committed

Two factors are involved in the calculation of SF. The first factor is the degree to which security has been satisfied in the past, measured by the number of conflicts for which security has been maintained. Secondly, we also assume that the greater the difference in security levels between the transactions involved in the conflict, the more important it is to maintain security. DMF is determined only by the number of deadline misses in the past. Note that for a comparison with DMF, (1 – SF) has to be used, since (1 – SF) is a measure of the degree to which security has been violated. Now, a simple comparison (1 – SF) > DMF is not enough, since different systems need to maintain different levels of security. Therefore, we define two weighting factors, $a$ and $b$ for (1 – SF) and DMF respectively. If $a \times (1 - SF) > b \times DMF$, then for the conflict under consideration, the security properties are more important and therefore the conflict is decided in favor of the transaction at a lower access class. If the opposite is true, then the transaction with higher priority is given precedence. Note that at low conflict rates, it may be possible to satisfy both security and real-time requirements simultaneously. As a result, the comparison is not made until the DMF reaches a certain threshold value DMISS_THRESH. The parameters DMISS_THRESH, $a$ and $b$ can be tuned for the desired level of security. A very high value of DMISS_THRESH or a very high value of $a$ compared to $b$ would result in SF being maintained at 1.0, i.e., for all conflicts the security properties are satisfied. A very high value of $b$ compared to $a$ would result in an SF value of 0.0, i.e., the behavior would be similar to that of 2PL-HP [ABB92]. For a desired value of SF between 0 and 1, the values of $a$, $b$, and DMISS_THRESH would have to be tuned based on the arrival rate of transactions.

The hybrid protocol can be specified as follows:

If a conflict between a lock holding transaction T1 and a lock requesting transaction T2 arises, the conflict is settled using the following rules:

If DMF < DMISS_THRESH then follow the steps taken by the Secure 2PL protocol.
Else if $a \times (1 - SF) > b \times DMF$, follow the steps taken by the Secure 2PL protocol.
Else break the conflict in favor of the transaction with the higher priority.

## 4. Concluding Remarks

Many DoD systems tend to be large, complex, distributed, real-time, fault-tolerant, and require multilevel security. Often, supporting more than one different types of requirements is a challenging task, since some of them are not compatible with each other. In this paper, we discussed issues on supporting the requirements of real-time and multilevel security in database systems. There are still several issues that need further investigation. Defining the capacity of covert channels and how to use it to control the trade-offs between timing and multilevel security requirements is one of them. We have started to evaluate the performace of secure concurrency control protocols and different methods for integrated support of real-time and security.

## REFERENCES


[AFSB83] Air Force Studies Board, 1983, Committee on Multilevel Data Management Security, *Multilevel Data Management Security,* National Academy Press.

[ABB92] Abbott, R. and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," ACM Trans. on Database Systems, vol. 17, no. 3, pp 513-560, Sept. 1992.

[DAVI93] David, R. and S. H. Son, "A Secure Two-Phase Locking Protocol," *12th IEEE Symposium on Reliable Distributed Systems,* Princeton, New Jersey, October 1993, pp 126-135.

[DENN87] Denning, D. E., et al., April 1987, "A Multilevel Relational Data Model," *Proceedings of the IEEE Symposium on Security and Privacy,* Oakland, CA, 1987.

[HUAN91] Huang, J., "Real-time Transaction Processing: Design, Implementation, and Performance Evaluation," *Ph.D. Thesis,* University of Massachusetts, Amherst, 1991.

[KIM93] Kim, Y. and S. H. Son, "An Approach Towards Predictable Real-Time Transaction Processing," *5th Euromicro Workshop on Real-Time Systems,* Oulu, Finland, June 1993.

[OOPS92] OOPSLA 92 Conference Workshop on Object-Oriented Approach and Real-time Applications, Vancouver, B.C., October 1992.

[SIGM88] Special Issues on Real-time Database Systems, ACM SIGMOD Record, March 1988.

[SHA91] Sha, L., R. Rajkumar, S. H. Son, and C. Chang, "A Real-Time Locking Protocol," *IEEE Transactions on Computers,* vol. 40, no. 7, July 1991, pp 793-800.

[SON90] Son, S. H., "Real-Time Database Systems: A New Challenge," *Data Engineering,* vol. 13, no. 4, Special Issue on Directions for Future Database Research and Development, December 1990, pp 51-57.

[SON91] Son, Y. Lin, and R. Cook, "Concurrency Control in Real-Time Database Systems," *Foundations of Real-Time Computing: Scheduling and Resource Management,* A. Van Tilborg and G. M. Koob (eds.), Kluwer Academic Publishers, 1991, pp 185-202.

[SON92] Son, S. H. and S. Koloumbis, "Replication Control for Distributed Real-Time Database Systems," *12th International Conference on Distributed Computing Systems,* Yokohama, Japan, June 1992, pp 144-151.

[SON92b] Son, S. H., J. Lee, and Y. Lin, "Hybrid Protocols using Dynamic Adjustment of Serialization Order for Real-Time Concurrency Control," *Journal of Real-Time Systems,* vol. 4, Sept. 1992, pp 269-276.

[SON93] Son, S. H., J. Lee, and H. Kang, "Approaches to Design of Real-Time Database Systems," *Database Systems for Next-Generation Applications - Principles and Practice,* W. Kim, Y. Kambayashi, and I. Paik (eds.), World Scientific Publishing, 1993, pp 120-131.

[SON93b] Son, S. H. and B. Thuraisingham, "Towards a Multilevel Secure Database Management System for Real-Time Applications," *IEEE Workshop on Real-Time Applications,* New York, New York, May 1993.

[SON94] Son, S. H. and S. Park, "Scheduling Transactions for Distributed Time-Critical Applications," *Distributed Computing Systems,* T. Casavant and M. Singhal (eds.), IEEE Computer Society, 1994.

[STAC90] Stachour, P., and B. Thuraisingham, June 1990, "Design of LDV - A Multilevel Secure Database Management System," *IEEE Transaction on Knowledge and Data Engineering,* Vol. 2, #2, 1990.

[THUR91] Thuraisingham, B., "Multilevel Secure Object-Oriented Data Model: Issues on Noncomposite Objects, Composite Objects, and Versioning," *Journal of Object-oriented Programming,* vol. 4, 1991 (also published in The MITRE Journal 1992).

[THUR92] Thuraisingham, B., "A Note on the Security Impact on Real-time Database Systems," *5th Rome Laboratory Workshop in Database Security,* Fredonia, NY, October 1992.

# Tradeoff Areas for Secure System Development

Catherine Meadows
Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC 20375

phone: 202-767-3490
fax: 202-404-7942
email: meadows@itd.nrl.navy.mil

# Tradeoff Areas for Secure System Development

Catherine Meadows
Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC 20375

**Abstract**

In this paper we identify several areas in which the satisfaction of security requirements can affect the cost and performance of a system, and describe what is known about tradeoffs in these areas. The areas we investigate include both features offered by the system and the procedures that are involved in building the system.

## 1 Introduction

When designing a system of any magnitude, it is necessary to keep in mind that it must satisfy a large number of requirements, some of which may be in conflict with each other. In some cases it may be necessary to trade off the different requirements against each other; in other cases it may be possible to identify other system features that may be traded off to resolve the conflicts between the two requirements. This is as true of security as it is of any other system requirement.

In order to make it possible to identify security tradeoffs it is necessary first to identify the various areas in which security can affect the cost or performance of a system, and then to identify the ways the various costs can be traded off, either against each other or against other system requirements. The purpose of this paper is to identify such a set of potential tradeoff areas security that can be used as a basis for for a more detailed investigation in the future.

The rest of the paper is devoted to a discussion of four major tradeoff areas that we have identified. The first two are in the area of communications, where by "communication" we mean any method of data transfer, including reading files, writing to files, and sending of messages over a network. There are two potential costs here: costs arising from restrictions on communications, and costs arising from additional required communications. Restrictions on communications are necessary to maintain secrecy and integrity of data; these restrictions may have an impact on system performance, since communications that are normally used in operating a system may be prohibited or restricted. On the other hand, additional communications may be necessary in order to authenticate different parts of a distributed system to each other and to exchange information, such as cryptographic keys, that are necessary to the preservation of privacy and integrity of communications.

The other two areas we examine are less easy to quantify. These are areas that are relevant more to the procedure of building or certifying a secure system than to the system features themselves. They are the amount of time it takes to evaluate and certify a secure system, and the risk posed by using the system. Risk and assurance can be traded off against each other, but, as we shall see, there are other tradeoffs possible when this one is not desirable.

## 2 Restrictions on Communications

Restrictions on communication can range from the obvious to the esoteric. One the more obvious end of the spectrum, for example, it is clear that the ability to modify software that is involved in running an operating system should be greatly restricted. It is precisely the lack of such restrictions that make most personal computers so vulnerable to viruses, for example. For systems that are supposed to offer an extremely high degree of data protection, such as multilevel systems which are intended to protect data classified at different security levels, the requirements can be much more stringent. According to the Trusted Computing System Evaluation Criteria (TCSEC) [DoD83], code must be divided into the Trusted Computing Base, which is charged with enforcing the security policy, and untrusted code. The untrusted code usually constrained by some form of the Bell/LaPadula model [BL76], in which an untrusted process is assigned a security level and is allowed only to read data at its level and below, and only to write data at its level or above. The reason for the latter is because the untrusted process is not trusted not to copy highly sensitive data down to a less sensitive repository. This "no write-down" requirement may be further restricted for integrity reasons.

Restrictions on communications in multilevel systems become even more strict as the data protected becomes more sensitive and the risks posed leakage or disclosure become greater. At this point, it also becomes necessary to guard against the exploitation of covert channels. These are communication channels be means of which an untrusted process running at a high security level can pass sensitive data to an untrusted process at a low security level making use of an effect the high process has on the system that is visible to the low process. Give an example here.

The easiest way of closing a covert channel is to partition all system resources. This of course has a very negative effect on system performance. Thus other methods have been developed, such as adding noise to a channel by making a shared resource unavailable from time to time, that can have variable effects on system performance, depending on the degree to which one wants to reduce the capacity of the channel. A number of studies documenting the tradeoffs between performance and capacity reduction have appeared in the literature. These include empirical studies that measure the results of applying various techniques on implemented channels [BCG+94], and studies that use information theory to compute the capacity of channels that may be affected by a number of different parameters [Mil87, TG88, Mos91, Gra93].

## 3 Required Communications

If a system is distributed, so that some communication between components takes place in a hostile environment that may include passive or active eavesdroppers, than additional communications may be required in order to authenticate various components of the system to each other and to distribute cryptographic keys so that components can communicate securely. Traditional low-overhead solutions such as passwords do not provide sufficient protection in such an environment; an intruder can simply read the password as it goes across the communication channel and use it to gain access to the system. In order to gain even a reasonable degree of security, some use of encryption, both for secrecy and authentication, is required. This means that protocols for key distribution and authentication need to be introduced that can increase the burden on a system in a number of ways.

In general, there are three ways in which the introduction of cryptographic protocols for authentication and secrecy can affect system performance. One of these is the number or length of messages. A mutual authentication protocol involves at least three messages, for example. A key distribution protocol will require five or six, or more depending upon the application involved. The

second added cost is the expense of performing the encryption operation itself. The third added cost is the overhead required to manage, guard, and generate cryptographic keys and authentication information. A cryptographic authentication system must make the use of one or more authentication servers whose job it is to provide the information that parties using the system need to authenticate each other, as well possible to generate keys that will be used for secure communication.

Tradeoffs between security and performance in this area are not necessarily clear-cut, but in general there is some correlation between the number of messages used in an authentication protocol and the degree of security that is achieved. In [Gon93] a study of several kinds of protocols describing the degree of security of each kind together with the number of messages needed to achieve it makes this tradeoff explicit.

It is possible to trade off the various performance costs of secure communication against each other in some cases. The use of public key cryptography can greatly reduce the amount of management overhead needed. When single key cryptography is used, the authentication server must know a master key for each principal, which must be kept secret. Any response to a request to have a session key delivered to a principal must be responded to by a message encrypted with that principal's master key. On the other hand, if public key cryptography is used, only the principals' public keys, which do not need to be kept secret, need to be known. It is possible even for the authentication server to go "off-line" by providing each principal a certificate signed with the server's private key that contains that principal's name and public key as well as any other relevant information. However, public-key cryptography is usually more computationally expensive than single-key, enough so that performance can be visibly effected by its use. Thus in this case we are trading off performance costs against key management overhead costs.

# 4 System Development and Evaluation Time

System development and evaluation time is the greatest hidden cost of a secure system. In order for a secure system to be usable as such, it must not only be secure, it must be believed to be secure. But often the work required to provide such assurance can add a significant to the development time and expense of a system. The extended time necessary for evaluation can have a hidden negative effect on system performance; often system performance suffers not as a direct result of implementing security mechanisms but because the system evaluation takes such a long time that the latest means for improving system performance are not used, simply because they were not available at the time the system was built.

There are several strategies for reducing the amount of labor involved in providing assurance. One, of course, is to keep the size of the part of the system charged with enforcing the security policy as small and well-structured as possible. This is a well-known principle, that is used as the basis of the TCSEC. Another, perhaps less well-known strategy, is to limit the kinds of secure systems that can be developed. The reasoning behind this strategy is that, if only certain kinds of systems can be developed, then the techniques for securing them and assuring that the security is adequate will become well enough understood so that they can be applied in a timely fashion. This, for example, was the strategy behind the TCSEC, to only allow a small number of classes of secure systems that could however be used in a wide variety of applications [Pot94]. This approach of course has the danger that the limits may be so severe that it is not possible to build compliant systems that enforce necessary security requirements. This indeed is a complaint that is often made against the TCSEC; thus the newer criteria, such as the European ITSEC [ITS91] generally provide more flexibility for the designer. However, this greater flexibility has the potential of increasing the amount of work needed to provide assurance; decisions that were hardwired in the more restricted

approach now must be justified.

## 5 Risk

Risk is a function of the degree of harm that will result if a system fails to perform its security functions properly and the likelyhood that such a failure will occur. One way of reducing risk is to increase system assurance; however, there are other approaches that may also reduce risk that can be applied. One of these is to reduce the amount of functionality provided by the system. In general, the more functionality a system provides, the greater the security risk will be. Greater functionality gives the user more power to perform actions that may be harmful, and it also may introduce added complexity which may make it harder to assure that a system performs its security functions properly. To give some examples, consider an ATM system which users can only use to find out the size of their bank accounts versus one in which users can also withdraw cash and transfer money from one account to another [LL85], or consider a system which contains data classified at only one security level versus a system that protects data classified at different security levels.

The degree to which risk can be traded off against functionality is not that well understood, but in some special cases rules of thumb have been worked out. In general, it is understood that, the less functionality that a system offers, the less security functionality and assurance is required. This is the idea behind much of the work on risk analysis for security, and it has also been codified for use in deciding what TCSEC rating is necessary for a system. For example DoD Directive 5200.28 [DoD88] provides guidelines for the TCSEC rating required based on the range of security levels of data, and the range of user clearances; the greater the range of security classifications or clearances, the higher the TCSEC rating required. In other words, the greater the security functionality required, the greater the degree of assurance required. The work of Landwehr and Lubbes [LL85] takes this approach even further by including risk factors such as the local processing capability available to the user (e.g., programmable terminals versus fixed-function interactive terminals), the type of communication paths, and the capabilities the system gives to the user (e.g., transaction processing versus full programming). All of these risk factors involve adding more functionality to a system. Thus in this case the tradeoffs are between functionality and risk. When risk is increased the increased cost appears in the necessity for greater system assurance.

## 6 Conclusion

In this paper we have identified a number of tradeoff areas in system security that can be used as a basis for more detailed investigations. As we have seen tradeoffs can occur not only in the features offered in the completed system, but in the procedures used in providing the assurance that the system satisfies the desired security properties. It is necessary to investigate both before we can reach a full understanding of the tradeoffs involved in developing a secure system.

## References

[BCG+94] P. K. Boucher, R. K. Clark, I. B. Greenberg, E. D. Jensen, and D. M. Wells. Toward a Multilevel-Seucre, Best-Effort Real-Time Scheduler. In *Proceedings of DCCA4*, pages 33–45. January 1994.

[BL76] D. E. Bell and L. LaPadula. Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report MTR-2997, MITRE Corporation, March 1976. Available as NTIS AD A023 588.

[DoD83] Department of Defense Trusted System Evaluation Criteria. Technical Report CSC-STD-001-83, Department of Defense Computer Security Center, August 15 1983.

[DoD88] Security Requirements for Automated Information Systems. DoD Directive 5200.28, March 21 1988.

[Gon93] Li Gong. Lower Bounds on Messages and Rounds for Network Authentication Protocols. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 26–37. Association for Computing Machinery, November 1993.

[Gra93] J. W. Gray. On Analyzing the Bus-Contention Channel under Fuzzy Time. In *Proceedings of the Computer Security Workshop VI.* IEEE Computer Society Press, June 1993.

[ITS91] Information Technology Security Evaluation Criteria (ITSEC): Provisional Harmonized Criteria. Document COM(90)314, Office for the Official Publications of the European Communities, June 1991.

[LL85] Carl E. Landwehr and H. O. Lubbes. An Approach to Determining Computer Security Requirements for Navy Systems. NRL Report 8897, Naval Research Laboratory, May 13 1985.

[Mil87] J. K. Millen. Covert Channel Capacity. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, April 1987.

[Mos91] Ira Moskowitz. Variable Noise Effects Upon a Simple Timing Channel. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, May 1991.

[Pot94] Garrell Pottinger. Proof Requirements in the Orange Book: Origins, Implementation, and Implications. Technical report, Mathematical Sciences Institute, Cornell University, Feb. 11 1994.

[TG88] C.-R. Tsai and V. D. Gligor. A Bandwidth Computation Model for Covert Storing Channels and its Application. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, April 1988.

# Myths About Dependability in Complex Systems

Michelle McElvany Hugue
Trident Systems, Inc.
meesh@nemo.cs.umd.edu

**Abstract**

*Researchers and practitioners have become proficient at designing protocols and systems that achieve the timeliness requirements associated with real-time applications, such as task deadlines and responsiveness. The concept of complex system dependability, which is a broad term addressing the system ability to deliver the correct services in the absence of total system failure, or catastrophic faults, has not been fully integrated into this problem due to myths and misunderstandings about faults, their effects, their occurrence probabilities, and the types of faults which need to be tolerated. Faults are often assumed to be random, hardware and host failure, and not due to the application software, protocols, or application specific factors. A pervasive misconception about real-time applications is that dependability assurance in the presence of faults is either unnecessary or too expensive. In this paper, we address this and other fallacies about the role of effective fault handling in ensuring dependable system services.*

## 1 Introduction

During the last 25 years, technological advances have outpaced the development of system engineering, synthesis and analysis techniques that exploit and realize new concepts and capabilities in military or commercial products. The ECS Block Program has tried to remedy this situation by developing methodologies and prototype tools to support every stage in the lifecycle of a complex system. However, the theoretical justification for interpreting requirements, for making design and implementation decisions, and for assuring that the completed system matches both the requirements and the intentions of the requirements writers has yet to be developed completely. The IEEE has taken steps to address this problem by creating the TSC (technical segment committee) on (ECCS Engineering Complex Computing Systems) from researchers and practitioners in real-time systems, distributed systems, fault-tolerant systems, and related disciplines. However, a continuing bottleneck in the ECCS process is the lack of a common language among the different areas of interest.

The most successful effort to date has been that of IFIP working group 10.4, which has produced the concept of dependability as a general term to address key characteristics of a given system, described by editor J. Claude LaPrie in [1], the principle source of this note. Fortunately, dependability has become the latest buzzword for addressing the behavior of a system in the presence of faults, used as synonymous with adjectives like "fault tolerant", and "reliable" as system descriptors. Unfortunately a pervading misconception is that dependability is of concern only in mission- or life-critical systems needing high reliability, and is unnecessary in most real-time systems that focus on availability instead. Even worse, many believe that the concept of dependability does not embrace timeliness as an essential system quality. The fault removal or tolerance techniques needed to enhance system dependability are often discarded as too expensive or too restrictive for performance rated systems. Faults are assumed to be random, due to hardware and host failure, not due to the application software, protocols, or application specific factors. In this talk, we address these fallacies and other myths about dependability, its use throughout the system life-cycle, and the role of effective fault management in ensuring dependable real-time services.

## 2 Dependability--A Synonym for Reliability or Fault Tolerance?

**Dependability** is that quality of the system which justifies the trust placed upon the service it delivers [2]. The service delivered by a system is judged from the user's perspective, where a user is a human being or another system. The **impairments** to dependability are **faults**, that cause **errors**, which, if unchecked or ignored can lead to system **failure**. This definition is recursive in the sense that the failure of one system is commonly interpreted as a fault by another system which requires the services of the first system. The **means** to **procure** dependability include **fault tolerance** and **fault prevention** or **avoidance.** The **means** to **validate** or **assure** dependability include **fault removal** and **fault forecasting** or **prediction.** The important attributes

or measures of dependability include **reliability, availability, safety, security.** Thus, dependability is not a replacement for any single term that describes the quality of service provided by the system. In fact, any measure of system service quality, such as **responsiveness** in a real-time system, or **atomicity** in a communication protocol, are measures of dependability. Some level of dependability is an essential characteristic of any system whose services must be trusted, whether or not faults have occurred.

## 3 Must real-time systems be dependable?

Many developers and users of real-time systems believe that fault-tolerance, and therefore dependability, are overkill for their systems. In fact, the definition of correct service inherent in the dependability definition also applies to timeliness and response time. That is, a system which fails to meet all its deadlines is by definition not dependable. A surprising aspect of this misconception is that the same designers see no problem in implementing checksums, parity, error checking and detection codes, and handshaking in communications, all of which are designed to either detect or mask faults.

## 4 Is dependability an all-or-nothing prospect?

Too often I have heard colleagues assess the fault resiliency of a system by saying that because of a **single point of failure,** the system was not fault tolerant, and therefore not dependable. This is also the myth of the fault-free system. Both system specifiers, implementers and users commonly require, claim, or assume that 100% fault detection or coverage is necessary and possible in a real-time computing. In point of fact, the concept of dependability does not require a system to be resilient to all possible faults. More practically, the potential impairments to dependability in a specific system need to be stated as to fault type and number. For example, a triplicated voting circuit is guaranteed to achieve the correct value as long as any two of its inputs are correct in both time and value. While the circuit is not capable of tolerating two incorrect inputs, it remains resilient to some subset of all faults.

## 5 Can't all system host failure semantics be predicted?

Work on handling transient system overloads and assuring system responsiveness abound in the literature. However, many of these approaches neglect the need for dependability in real-time applications. Some assume a tiered approach to providing system services, in which the host and its operating system are the lowest layer, followed by a fault-tolerant service layer. The actual system services are at the highest level in the tier. In models of this nature, the operating system is the weakest link. Not only are host failures assumed to be random, but the fault-tolerant services are designed based on assumptions about the failure semantics of the host. If these assumptions are violated, then the system fails and rollback or retry after the host recovers is needed to restore the portion of the real-time application assigned to the failed host. Since most operating systems are inherently unpredictable, we have the fault-tolerance layer essential to ensuring correct operations in the presence of failures relying upon a system whose fault resiliency is questionable at best. If the assumed failure semantics are violated, then the fault-tolerance protocols may fail without the applications layer being aware of any problem. The result could be an incorrect output or decision which cannot be identified as such.

## 6 Discussion

This brief note has addressed but a few of the myth-conceptions about dependability and the role of faults and fault tolerance techniques in assuring dependability. The suggestions provided regarding the inclusion of effective fault-handling methodologies suitable for real-time and dependability are far from sufficient to address these issues. Currently, the development of dependable complex systems remains an art, not a science. However, the adoption of common terminology among researchers and practitioners is the first step in providing a common framework for system specification, design, development, and testing. Interested readers should contact the author regarding participation in the ECCS, to continue to promulgate this work and its approach to complex computing system design and usage.

## Acknowledgements

The author wishes to thank the Office of Naval Research for their continued support of research in this area, currently under Navy contract N00014-94-C-0085 and through the ECS Technology Block Plan.

## References

[1] Jean Claude Laprie, *Dependability: Basic Concepts and Terminology.* Springer-Verlag, 1992.

[2] W. C. Carter, "A time for reflection", in *Proceedings of 8th IEEE International Symposium on Fault Tolerant Computing (FTCS-12),* Santa Monica, CA, June 1982, pp. 41.

# A Step Toward Integrating Dependability Concerns into the Systems Engineering Process

Richard C. Scalzo
Naval Surface Warfare Center Dahlgren Division
10901 New Hampshire Avenue
Silver Spring, Maryland 20903

**Abstract**

There are two problems which make systems engineering for dependability difficult. The first is that methods for developing dependable systems are well-known but are not yet well-integrated into the systems engineering process. The second problem is related to the first. It is that information capture for dependability is incomplete. This paper focuses on the problem of capturing dependability information relating to system interfaces and observability, and the propagation of errors and exceptions. Background for the problem is given in Section 1. Sections 2 and 3 contain the identification and a proposed classification of information on observability, errors and exceptions. Finally, in Section 4, a mapping of the information to meta-models for data interchange and design capture is contained in the last section.

## 1 Introduction

The systems engineering process for real-time systems usually leaves dependability concerns to be treated in an ad hoc manner after other system design factors, such as those related to performance, have been addressed. Thus trade-offs for dependability are delayed until late in the design process. This has adverse effects on the cost of a system and on its development schedule. In order to solve this problem, an improved understanding of the semantics underlying dependability concepts is needed. Much work in formulating the semantics for dependability has been done and is now mature enough to support the development of systematic systems engineering techniques for dependability. These semantics can be used to develop meta-models for the support of the design process and for the support of data-interchange. A meta-model is a model which describes a class of models, and is used to generate models. The meta-models for data-interchange are needed to support the system design and evaluation processes.

If dependability considerations are to be integrated into the system design process there are several questions to be answered:

a. What dependability information needs to be captured?
b. Which meta-model should it be assigned to?
c. How should it be represented?

This paper attempts to give partial answers to these questions. Dependability terminology is listed in Section 1.1 in order to introduce the basic concepts. In Section 1.2 the design views and meta-models needed for systems representation are discussed. A similar discussion appears in Section 1.3 for data-interchange.

### 1.1 Terminology

A great deal of work is going on in the computer engineering and standards communities dealing with issues of dependability terminology, and with frameworks for developing dependable systems. One such effort is the ongoing work by the International Federation for Information Processing (IFIP) 10.4 Working Group. The work of this group has been published as [Laprie 92], and is the source of most of the definitions provided below. Note also that the definitions may be extended to include systems with mechanical subsystems and humans in them. In all of the definitions below, it is useful to think of a system represented in a client-server form.

(1) A system is a collection of parts called components which interact with one another under the control of an interaction design. The concepts of system and component are hierarchical in nature, i.e., a system or a component may contain other systems or components.

(2) Dependability is defined to be the "trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers."

(3) Service is defined in terms of system behavior as it is perceived by its users.

(4) A user is another system which interacts with the given system.

(5) The specification of a system describes the user's expectations in terms of its functionality, services to be delivered, and the conditions under which these expectations are to be met. The specification describes what should happen when the system is operating, as well as what should not happen. This leads to the specification of additional functions and services of what the system should do to lower the likelihood of occurrence of what should not happen. The system specification is assumed to be authoritative, i.e., complete and consistent, and agreed upon by the builder and the receiver of the system.

(6) A failure occurs when the service delivered is not in compliance with the system specification.

(7) An error is that part of the system state which may lead to a system failure. Moreover, an error which affects the service that a system delivers is a clear indication that a failure (at some level) has occurred. In this sense an error is a manifestation of a fault, where

(8) A fault is defined to be the known or hypothesized cause of a failure.

(9) The environment of a system is defined to be other systems which interact with or interfere with the given system.

(10) The function of a system is defined to be what the system is intended to do, while,

(11) The behavior of a system is what it does, and

(12) The structure of a system is what makes the system do what it does.

(13) The components at the lowest level of a decomposition are referred to as atomic, as in [Lee 89]. The level of decomposition required to adequately represent a system depends on the purpose of the representation, as well as the type of representation chosen.

(14) The state of a system is defined as a "condition of being with respect to a set of circumstances, whether of behavior or of structure," [Laprie 92]. Following [Lee 89], it may be useful to think of representing system behavior as a sequence of external states of a system, i.e., states which appear at the boundary of the system with its environment. In this representation scheme, the external behavior of a system is described in terms of external states. Thus, a system undergoes transitions from one external state to another. The internal states of a system are defined as the external states of its components.

(15) An interface is a place at which two systems interact, [Lee 89]. This definition is meant to be an abstraction of the usual definition associated with hardware. A fundamental distinction needs to be made at this point. The definition of an interface as a hardware component which connects two devices will not serve the purposes of this paper, since such a component should be classified as another system for the purposes of system description. Hardware interfaces, such as data busses, will be referred to as a hardware realization of a system interface. It is by means of interfaces that, a system interacts with its environment.

Since dependability is defined in terms of the behavior of a system, much of the information which needs to be captured is related to the behavior of the system. Thus, it is important to capture information

for high-level system states. Since the state of a systems is dependent on its structure, it is also important to capture this concept. It should be noted that the structure of systems changes over time. This means that the concept of system state should include information about the structure, as well as time related information.

## 1.2 Dependability and Design Views

Systems engineering information needs are qualitatively and quantitatively different from the information required by engineers engaged in the detailed design. In addition, there are many ways in which the design of systems can be represented, although none of the representations which are in common use currently capture all of the information required for the design of a dependable system. Each design view gives a different perspective of a system. A set of design views should have the property of allowing a complete and consistent representation of a system. In addition, a set of design views should partition the system design into the important perspectives of system design. For each design view one or more methods can be used to represent a system design, [Hoang 91]. In this section, the various views required for capture of design information are discussed.

A set of five design views are summarized below. It is felt that these views can capture the design information required by systems engineers. Some of these design views are commonly used in systems engineering and design and are widely available in CASE tools. The views described here are the environmental view, the informational view, the functional view, the behavioral view, and the implementation view. These views, the purposes and the design elements associated with them are depicted in Figure 1 below.

Associated with each of the views is one or more design representation methods. Each of them may have one or more meta-models that may be used to generate the representation. These meta-models are intended to be capable of generating CASE tool models for system representations. Since these meta-models vary among CASE tool manufacturers, standards for them are under development. By using a standard meta-model for system representation, all of the relevant information can be captured for a given system development activity. It is assumed that the meta-models considered in this paper adhere to the Electronics Industry Association Case Data Interchange Format (EIA CDIF), whenever this standard is applicable, see [EIA 1] - [EIA 3] for details.

Some models for descriptions of a system may be integrated into a logical model. This conforms to structured analysis and design techniques currently in use. The main components of a logical model are a data-flow model, a control-flow model, a data model, and a state-event model. Meta-models for these important parts of design representation have been developed or are undergoing development in industry, e.g., [EIA 1], and as part of the Engineering of Complex Systems (ECS) Program. These meta-models may then be used to build models for system representation. The design views shown in Figure 1 go beyond this. These views include information on the global environment for the system, as well as the resources that make up the system. Nonfunctional system attributes must also be included in the design representation models. This type of information is included in the environmental view and the informational view.

Nonfunctional attributes, such as system availability, are expressed as system design factors. These factors are attributes that are attached to design elements. The list of design factors identified to date contains 100 system-level attributes, [Nguyen 94]. The design factors are organized into a hierarchy that supports the design process, as required by systems engineers. Dependability is associated with a set of nonfunctional attributes. As of now, some dependability information is included as a class within this hierarchy. Moreover, many of the design factor classes also contain elements which may be classified as dependability information. These design factors define elementary dependability metrics from which system-level dependability metrics may be built. Other design factors currently have no metrics but are important because, when refined, they will allow the systems engineer to express other measures of robustness, such as the number of faults of a given class that a system can tolerate. An important unfinished problem is to identify possible additions to the hierarchy of system design factors. Although some system design factors

| DESIGN VIEW | VIEW OBJECTIVES | DESIGN ELEMENTS |
|---|---|---|
| Environmental View | (1) Establish Conditions and Events Which Constrain System Operations<br>(2) Specify Performance MOEs and Conditions of Measurement | (1) Environmental Conditions and Event Descriptions<br>(2) Design Requirements and Constraints<br>(3) System Initial Conditions<br>(4) MOEs |
| Informational View | (1) Characterize System Concept of Operations<br>(2) Represent System Components in Abstract Terms | (1) Entity, Relationship, Attribute Diagrams<br>(2) Attribute / Method Descriptions |
| Functional View | (1) Define System Functions and Decompose Them<br>(2) Specify Data Flow Requirements | (1) Function / Data Flow Diagrams<br>(2) Process Specifications<br>(3) Data Dictionary |
| Behavioral View | (1) Define System States and Triggers<br>(2) Specify System Behavior Characteristics | (1) Control Flow Diagrams<br>(2) State Transition Diagrams<br>(3) Control Specifications |
| Implementation View | (1) Define Physical Hardware, Software, and Human Resources Which Make up the System<br>(2) Specify System Physical Interconnections | (1) Hardware, Software, and Human Resource Descriptions<br>(2) Performance Parameters and Resource Characteristics<br>(3) Function-Resource Mappings |

**Figure 1. Five Design Views**

associated with dependability will be associated with meta-models for system design, some will be used in system evaluation and assessment. Each factor associated with system evaluation should be mapped to a meta-model.

1.3 Dependability and Data Interchange

For any integrated set of system engineering tools, the capture of design information and the abstract concepts of design views and their meta-models are related to data interchange meta-models. These relations arise from the need to pass information among various tools for systems engineering. There are tools for top-level requirements and requirements traceability, tools for system evaluation, and tools for system optimization to be considered. To support data interchange among them, the CDIF meta-models cited above have been developed, [EIA 1] - [EIA 3]. In the area of design capture, meta-models support data interchange between models for data flow diagrams, state-event models, and models for design implementation. Another meta-model to support data interchange for nonfunctional system design factors is under development. Most of the dependability data not currently incorporated in the meta-models for data interchange can be captured by adding appropriate sets of attributes to these models. These attributes can then be mapped onto meta-models for design capture, system evaluation, system assessment and requirements traceability. A top-level diagram indicating the main paths of data interchange for the systems engineering process is shown below as Figure 2.

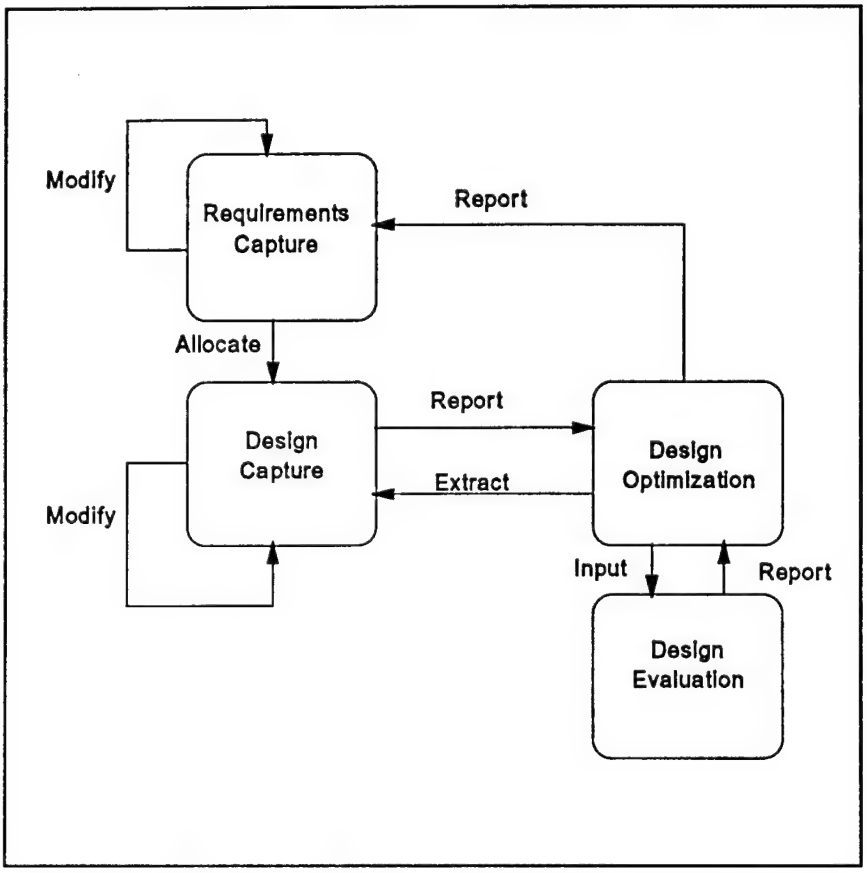



**Figure 2. Data Interchange Paths**

## 2 System Description Concepts

Since every system can be represented as a hierarchy of interacting components, a representation of these components and their interactions must be captured for design and evaluation. Also components in a system interact at interfaces, so that a proper specification of interfaces and the kinds of interactions that can occur is required. A discussion of these ideas appears below. This is followed by a discussion on observability of system behavior.

### 2.1 System Components, Services and Interfaces

The dependability specification of a system is not complete unless it accounts for system behavior. The behavior of a system is defined as what a system does. More formally, a system is said to interact with its environment and to respond to stimuli at the interface between the system and its environment, [Lee 89]. Following [Lee 89] further, the states of a system may be described as external and internal. The external state is defined by the state associated with the interface of the systems with its environment, while the internal state of a system is defined to be the aggregate of external states of the components within the system. At a given level of decomposition, the internal state of a system may not be directly visible at the interface with its environment, depending on the information structure of the system. Information on the internal state is filtered through the external interface of the system. A more detailed definition of behavior

is to define it to be the sequence of states that appear at the interface of the system with its environment. Finally, note that these definitions are applicable to hardware, software and systems when operating. The engineer interested in the behavior of a system can only observe that behavior as an abstraction at an external interface for that system, unless the level of decomposition of the system makes them accessible. These ideas are elaborated on in the next section.

## 2.2 Dependability and Observability

A systems engineer needs to know the high-level location of points in a system at which its behavior is observable. The reason this is necessary is that dependability can only be understood in terms of the observed behavior. Thus, it is important to distinguish between interfaces at which it is not possible to observe the behavior of a system or one or more of its components. The term "instrumented" will be used to describe interfaces at which it is possible to observe the behavior of a system. However, even though an interface is instrumented does not mean that system behavior is observed there. Before the behavior of a system can be observed mechanisms for observation must be in place. An interface which has these mechanisms in place will be call "activated." Thus the behavior of a system is available for observation at an interface if it is instrumented and activated. If the information is actually collected then the interface will said to be "monitored". So we see that the behavior of a system is observed only when an interface is instrumented, activated, and monitored. This information is important while a system is under design, and when it is under test and evaluation.

Interfaces may also be classified by noting if they are always incorporated into the design of a component. Such interfaces are called standard interfaces. For example, a realization for one type of standard interface consists of the addressable registers in a computer processor. In other cases an interface may be optional in the sense that it is not always incorporated into the design of a component. An optional interface may be thought of as one that is put there for testing purposes, such as the interface which results when a connection for a probe in the backplane of a computer which is used only during system testing.

The system attributes that have been identified above must be incorporated into the information structures that support systems engineering. Thus the above classification scheme for interfaces must be incorporated into the meta-models for data interchange. The data interchange meta-models need this information, not only for inclusion as design data, but also because it is required in order to utilize certain types of system evaluation tools

# 3 System Components, Errors, and Exceptions

In this section, a treatment of errors and exceptions is undertaken because the specification of the dependability of a system is incomplete unless the system states have been partitioned into three classes: (1) correct , (2) incorrect and recoverable, and (3) incorrect and unrecoverable. It will be assumed that a partition of the states into the first two classes is all that is needed, since the third depends on these categories. With this in mind, a discussion of errors and exceptions follows.

The response a component presents to its environment through its external interface can be classified into two general categories: normal and abnormal, as depicted in Figure 3 below.

Since an internal state of a component is not directly observable, neither is its internal behavior; however, an abnormal response from a subcomponent may cause the component to generate an abnormal response. The term abnormal response is used because not all responses to requests for services can be classified as erroneous or correct. For example, consider a hard disk subsystem. Suppose that the requirement for this disk is that the average response time for a write to this disk be 14 ms or less, with a variance of 4 ms, and that the underlying distribution is Normal. Now if a particular write to the disk takes 34 ms, and after this response, the cumulative mean response time is 12 ms and the variance is 2 ms, the response might be classified as an abnormal response, which may indicate an impending failure of the disk, but which may still meet the requirement for the disk. In the interest of simplicity, it will be assumed that all

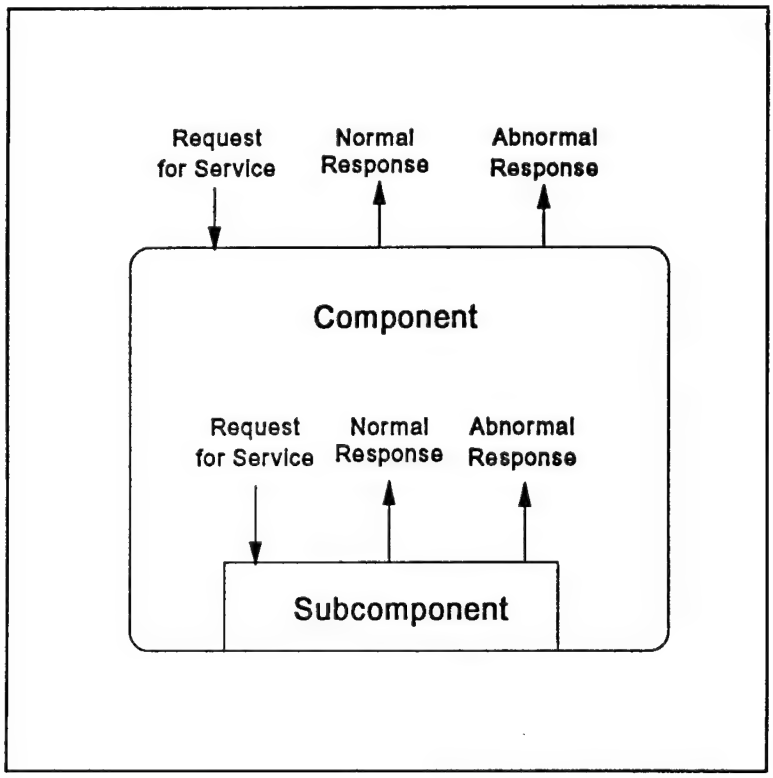



**Figure 3. Normal and Abnormal Responses**

abnormal responses are caused by errors in the state of the requesting component or the servicing component. This means that all abnormal responses will be identified with the occurrence of an error.

### 3.1 Errors and Exceptions

[Lee 89] defines an exception as the occurrence of an abnormal response, and although this term is used primarily for software systems, it may be generalized to include hardware and any algorithm running on hardware. Flaviu Cristian in [Cristian 93] defines a [software] exception as occurring when a program is invoked in its failure domain; i.e., it is invoked in that part of its set of possible inputs which will not lead to a normal termination. Either definition has the requirement that service a component delivers be abnormal in some sense. In view of the assumption above, every exception will be associated with the occurrence of an error. When an exception occurs, a component is said to signal it to notify the requesting component that the service cannot be delivered properly.

Exceptions that a component experiences may be classified as interface exceptions or failure exceptions. An interface exception occurs when a request for service is received and that request does not conform to the interface specification of the component. Typical examples of interface exceptions which may be signalled are arithmetic overflow, protection violation, and address violation. Interface exceptions may be caused by a design fault in the requesting component, but this is not necessarily the case. Exceptions associated with the presence of a fault in the system are called failure exceptions. For example, suppose a memory component contains a parity check, with the property that it can detect but not correct an error in stored data. In this case a fault has led to an error and it was detected, but all that the component is

capable of doing is to signal that an error was detected and that it was unable to deliver service as specified.

The discussion above classifies exceptions and the manner in which they are signalled. It does not address the question of what to do about exceptions. When an exception has been signalled, it is assumed that it has been detected but not corrected. If an exception has been detected, the next step is to determine what, if anything can be done to correct, or handle it. Figure 4 below depicts the classification of exceptions which appears in [Lee 89].

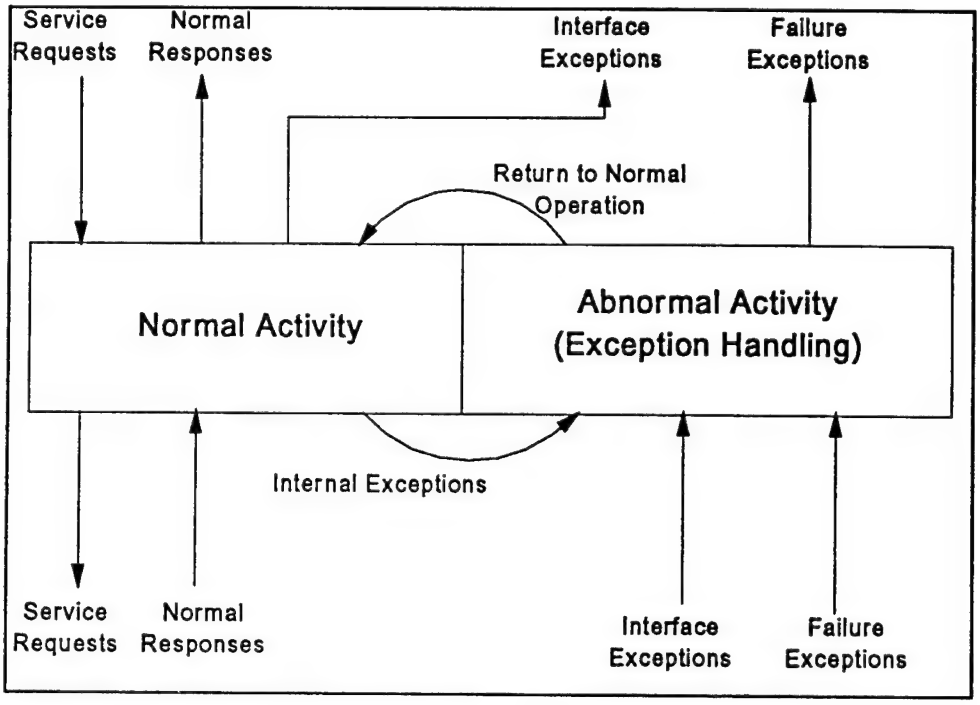



**Figure 4. Classification of Exceptions**

Exceptions as depicted in Figure 4 may be classified into two types, internal and external, according to whether they are handled within the component in which they occur or are handled by the component which requested the service. Exceptions that a component signals are classified as external exceptions, since the exception handling mechanisms are external to the component which signaled the exception. Those exceptions handled by mechanisms internal to the component are termed internal exceptions. In order to distinguish external from internal exceptions, internal exceptions are said to be raised, rather than signalled.

In addition, the above classification assumes that the components are ideal, i.e., that they detect and signal all external exceptions and they raise and handle all internal exceptions. No system can be made to behave in such an ideal fashion. So another category of exceptions is needed, viz., the category of exceptions which occur but are not raised or signalled. Design tools need to provide a means for classifying exceptions which are anticipated and handled by the design, as well as those which occur and are unanticipated, and as a result, not handled. This classification partitions exceptions into four disjoint sets: (1) Anticipated Internal, (2) Anticipated External, and (3) Unanticipated Internal, and (4) Unanticipated External. The anticipated categories can be used to trace the history and rationale for the exception handling mechanisms which are implemented in a design, for as a design is implemented this category will have additional items added to it from time-to-time.

3.2 Propagation of Errors

Recall the assumption that exceptions are considered to be triggered by errors in the system. This means that propagation of an exception is equivalent to the propagation of an error. Providing dependability in a system means that preventing propagation of errors is an important design goal.

Even when the design of a system is robust, certain types of failure exceptions cannot be handled in the component in which they occur, nor can they be handled in the component requesting the service. In [Cristian 93] such exceptions are called " unconfined." Examples of unconfined exceptions include the following: (1) a component terminates its service activity normally, yet it does not deliver the service as specified; (2) an exception is detected and is signalled to the component which requested the service; furthermore, the exception handlers in this component signal the occurrence of an exception to one higher in the hierarchy. In the second case, the exception signal has propagated. If this occurs and no corrective action is taken, then the error associated with this exception has propagated. As noted above exceptions and the errors associated with them may also propagate without detection.

In order for the error confinement regions of a system to be identified, it must be known when and where in the system an exception signal will be acted upon. If correction of the error associated with an exception occurs in a component with an external interface which is not observable, then the error associated with the exception is said to be masked. The only way for the system to detect such a masked error is to make certain that an exception signal is propagated to a component with an activated, instrumented interface, along with the notification that an exception has occurred and been handled. If the exception is not handled, then notification that the exception has occurred and that the error has been masked should be made. In this case, only those exceptions which are unconfined and the errors associated with them will propagate. Thus there is a need to identify with each interface, the errors which it is meant to be an error containment boundary for. Meta-models for design capture or data interchange should provide information for the level at which errors stop propagating, as well as the levels at which errors were detected or even masked.

4. Mapping of Information

The first three sections discussed the need to incorporate dependability information which is not currently captured in systems engineering tools. Two types of information were identified. One type deals with the observability of system behavior and how observability characteristics change over the life cycle of a system. The other type is used to mark interfaces among various components in order to make systems engineers aware of the major error containment regions of a given design. If this information is to be included in the design capture or data interchange meta-models, then a mapping of this information onto these meta-models must be done. The mapping can be made as attributes added to the existing meta-models or new relations.

The types of information that were identified and classified for inclusion in the data interchange meta-models and for possible inclusion in the design capture meta-models appears below , along with a proposed assignment to one of these meta-models as an attribute or relation.

(1) System State:
   a. Maximum Time Allowed in State ----> Attribute / Behavior Meta-Model
   b. Minimum Time Allowed in State ----> Attribute / Behavior Meta-Model
   c. Identifier for Component Up / Down ----> Attribute / Meta-Model

(2) Interfaces:
   a. Standard / Optional Identifier ----> Attribute / Implementation Meta-Model
   b. Identifier for Instrumentation ----> Attribute / Implementation Meta-Model
   c. Identifier for Activation ----> Attribute / Implementation Meta-Model

(3) Information at Interfaces:

a. Range Constraints Identifier ----> Attribute / Implementation Meta-Model
b. Data Type Constraints Identifier ----> Attribute / Implementation Meta-Model
e. Error Containment Identifier ----> Attribute / Implementation Model

(4) Exception / Error Information

a. Identifier of Scope of Exception Signalling ----> Relation / Implementation Meta-Model
b. Identification of Scope of Error Masking ----> Relation / Implementation Meta-Model
c. Identification of Exception Handling Mechanisms ----> Attribute / Functional Meta-Model or Attribute / Implementation Meta-Model

A top level meta-model for fault tolerance is under is currently under development. The purpose of this development effort is to help refine the identification of primitive concepts for fault tolerance. Once this is accomplished, the model may be used to express dependability information not currently captured in system design views. Finally, a proposed mapping of this information to meta-models for data interchange is being investigated.

## 5 References


[Anderson 89] **Dependability of Resilient Computers,** T. Anderson, (ed.), Blackwell Scientific Publications, UK, 1989.

[Cristian 1989] "The Role of Exception Handling in Dependable Programming," in **Dependability of Resilient Computers,** T. Anderson, (ed.), Blackwell Scientific Publications, UK, 1989.

[EIA 1] **EIA / IS - 106 CDIF - CASE Data Interchange Format - Overview,** Electronic Industries Association Engineering Department, 1994.

[EIA 2] **EIA / IS - 115 CDIF - Integrated Meta-model - Data Flow Model Subject Area,** Electronics Industries Association Engineering Department, 1994.

[EIA 3] **EIA / IS - 116 CDIF - Integrated Meta-model - State / Event Model Subject Area,** Electronics Industries Association Engineering Department, 1994.

[Hoang 91] **Mission Critical System Development: Design Views and Their Integration,** N Hoang and S. Howell, and N. Karangelen, NAVSWC TR-91-586, 1991.

[Kim 93] **A Real-Time Object Model: A Step Toward and Integrated Methodology for Engineering Complex Dependable Systems,** K. H. Kim and L. F. Bacellar, Proceedings of the 1993 Complex Systems Engineering Synthesis and Assessment Technology Workshop, July 1993.

[Laprie 92] **Dependability: Basic Concepts and Terminology,** J. C. Laprie, (ed.), Springer Verlag, 1992.

[Lee 89] **Fault Tolerance: Principles and Practice,** P. A. Lee and T Anderson, Springer Verlag 1989.

[Nguyen 94] **System Design Factors: The Essential Ingredients of System Design,** C. Nguyen and S. Howell, NAVSWCDD /TR-92/268, 1992.

# Testing and Fault Injection of Distributed Protocols

Scott Dawson and Farnam Jahanian

Real-Time Computing Laboratory
Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48109-2122

{sdawson,farnam}@eecs.umich.edu

**Abstract**

*A growing challenge confronting designers and implementors of safety-critical distributed systems is the evaluation and validation of dependability requirements. This paper address the problem of testing fault-tolerance capabilities of distributed protocols. It introduces a general framework for fault injection and testing of distributed systems and it describes an ongoing development of a tool based on the framework. The tool can be inserted between any two layers of a protocol stack, and it can be used to inject faults into the system by observing and manipulating messages that are exchanged between the two layers. Existing approaches to fault injection often handle memory and CPU faults. Most current approaches for testing distributed protocols do not allow the manipulation of the protocol into specific states since they depend primarily on random testing to obtain certain coverage. This makes testing of distributed protocols difficult because some states in the protocol are hard to reach simply by probabilistically dropping or delaying packets, or by randomly testing execution paths. We are evolving toward a method which will make it easier for the tester to manipulate protocols into hard to reach states during a test. Other features of this tool include the support for both probabilistic and deterministic testing of distributed protocols, user-defined test scripts that can guide the analysis at run-time, and executable specifications that can emulate a participant in a distributed protocol.*

*Keywords:* distributed systems, protocol testing, fault injection, executable specifications

## 1 Motivation

As the software for fault-tolerant real-time systems has become more complex, evaluating and validating system dependability is a growing challenge that confronts software designers and systems engineers. The shift from centralized computing towards collections of powerful and inexpensive microprocessors connected by communication networks has been a significant contributing factor to the complexity of these systems. Recent experiences with complex distributed systems such as the FAA air traffic control system and the Boeing 777 aircraft confirm that dependability evaluation and validation will become more crucial as large-scale distributed computing is becoming a reality in 1990s.

This work addresses the growing challenge being faced by the industry and government labs in ensuring that safety-critical distributed systems meet their prescribed specifications. This paper focuses on an integrated set of tools for specification, fault-injection, and testing of distributed real-time protocols. Numerous approaches have been proposed in the past for evaluating and validation of system dependability including formal methods, analytical modeling, and experimental methods. Techniques based on fault-injection have been proposed to test fault-tolerance capabilities of system. Hardware fault-injection [12, 6, 1] and simulation approaches for injecting hardware failures have received much attention in the past. Recent efforts have focused on software fault-injection by inserting faults into system memory to emulate errors [3, 11]. Others have emulated fault-injection into CPU components [9]. However, fault-injection and testing dependability of distributed systems has received very little attention until recently [5, 4, 2, 7]. Our work is motivated by several observations:

- In testing a distributed system, one may wish to coerce the system into certain states to ensure that specific paths are executed. This will require orchestrating a distributed computation.

- In testing the fault-tolerance capabilities of a distributed system, one often requires certain behavior from a participant in protocol that may be impossible to achieve under normal conditions. This may require the emulation of "misbehaving" participants by injecting faults into the system.
- Testing organizations will often require a testing methodology that does not instrument the code. This is particularly important for existing systems that have been operational for a long time and are being tested because of a modification.
- Most existing fault injection approaches depend heavily on probabilistic (or random) test generation. Due to the inherent complexity of distributed systems, one must provide the systems engineer with the necessary tools to guide the testing process.
- It is highly desirable to able to test the system using the requirement specification for a system. Furthermore, fault injection into a system that is partially implemented can potentially give valuable feedback to the system designer early during the development process.
- Injecting faults to test fault-tolerance mechanisms in a real-time system introduces new challenges that have not been addressed before. Timing faults often require the emulation of certain scheduling policies that ensure a particular behavior. Furthermore, one has to ensure that fault injection is as non-intrusive as possible on the real-time behavior of a system.

The main features of our fault injection and testing tool are summarized below:

- The tool can be inserted between any two layers of a protocol stack. This tool can be used to inject faults into the system in order to test the fault tolerance mechanisms of the protocol above the fault injector.
- The tool supports user-defined scripts that can guide the testing of a computation by observing and manipulating messages that are exchanged between two protocol layers in a stack.
- The tool supports both probabilistic and deterministic test generation. It also supports injecting various types of faults into a distributed system.
- The tool supports the inclusion of an executable specification to emulate the behavior of a participant in a distributed system. This allows the testing of a certain participant in a distributed system by mimicking bad behavior on the part of another participant which is running the protocol.
- The tool allows testing of a protocol without having to instrument the code of the protocol with test code.

# 2 Approach

We view a distributed protocol as a specification of a communication abstraction through which a collection of participants exchange a set of messages, much in the same spirit as the *x*-Kernel [8]. In this model, we make no distinction between application-level protocols, interprocess communication protocols, network protocols, or device layer protocols. As shown in Figure 1, each protocol is specified as part of a protocol stack. In the figure, both machines are running a set of protocols. Machine 2 is running a modified protocol stack, which contains a driver layer and a fault injection layer above and below layer 2. These layers can be placed around any layer in the protocol stack. The idea is that the layer which is being tested is between the driver and fault injection layers on the protocol stack. The driver and fault injection layers are used to manipulate the tested layer into certain states during the test.

The driver layer is responsible for generating messages and running the test. The fault injection layer intercepts all messages coming into and leaving the tested layer. It has the ability to drop, delay, or corrupt any packet which is sent or received. The driver and fault injection layers communicate with each other during the test and can coerce the system into certain states by generating, dropping, corrupting, or delaying messages. For example, if the fault injection layer needs to generate a spontaneous message, it asks the driver layer to generate a new message. It can then corrupt, delay, or drop the message and keep track of how the tested layer responds. The remainder of this section will discuss the three main features of this tool.

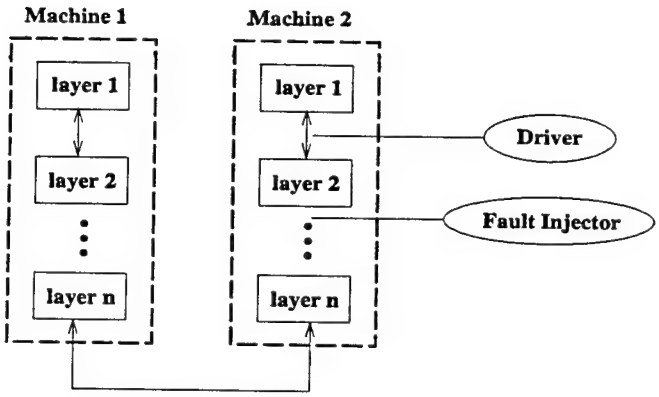



Figure 1: Protocol Stacks.

## 2.1 Script-Driven Fault Injection

For testing protocols which are already "out there" and are not instrumented with test code, a method is needed which does not require changes to the protocol being tested. This is accomplished as described above, with a fault injection and driver layer. The fault injection layer and driver layer are used to manipulate the protocol which is under test simply by sending messages through the protocol. Messages can also be corrupted, dropped, or delayed.

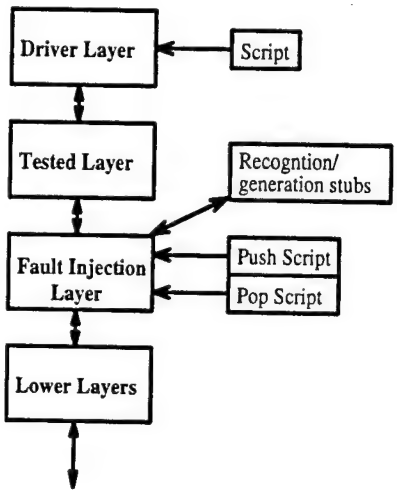



Figure 2: Script Interaction.

As seen in Figure 2, both the driver and FI (fault injection) layers run a script which controls their actions. The FI layer runs a "push" script each time a message is pushed (or sent down) the protocol stack. It runs a "pop" script each time a message is popped (or sent up) the protocol stack. These scripts have the ability to drop, delay, or corrupt messages. The scripts can specify certain distributions so that messages can be dropped probabilistically. The packet recognition/generation stubs shown in the figure are written by people who know the packet formats of the tested layer. An interface from the push/pop scripts to the stubs exists so that the scripts know what types of packets they are operating on. The scripts can use the packet generation stub to generate messages of certain types. This type of message generation can only be done if state of the tested layer doesn't have to be updated for the generated message. For example, when generating a spurious ACK message in TCP, no data structures need to be updated. The message can simply be generated and sent. However, when generating a data message in TCP, the sequence number would need to be used and updated, and the message would have to be kept track of in case retransmissions were needed. This type of message generation cannot be done by the fault injection layer because it does

not have access to the data structures of the tested layer.

Spontaneous actions can also be taken by the scripts. At startup time, a message with a special type called *SPONTANEOUS_TYPE* is generated and put into both the push and pop queues of the fault injection layer. When the push/pop script runs, the type is noted and desired actions can be taken. Typically, in addition to whatever other actions might be taken such as generating a spurious real message, another message of *SPONTANEOUS_TYPE* is generated and requeued with some random delay. In this way, spontaneous actions can continue to be taken during the course of the test.

The driver layer works as follows. It takes an executable specification of what messages it should produce and then generates these messages and sends them down the stack.

The reason for having a layer both above and below the tested layer is to allow creation of new messages and manipulation of messages generated by other participants in a protocol. A layer which only sits below the tested layer can drop, delay, and corrupt messages, but has a hard time generating messages because it cannot manipulate data structures in the tested layer. The driver layer is used for doing most message generation so that data structures in the code of the tested layer will be updated correctly.

## 2.2 Executable Specification

The ability to test a system before it is actually built can greatly reduce system development cost. Errors in protocol logic can be found quickly, before they would require more expensive code changes. If these errors can be corrected and the resulting protocol retested before a lot of code is actually written, fewer changes to protocol code will have to be made, thus making system development quicker and easier.

Executable specification allows the user to describe what types of actions certain modules in a system should take. In our specific case, executable specification means that the user can specify what actions a protocol layer should take when certain messages are received or sent. The user can give a high level description of what a part of the protocol should do, and the messages corresponding to this description will be generated. For example, if the user wants to design a database client, a fake server can be generated which doesn't actually look up any values when requests come in, but simply reply with "canned" answers to all queries.

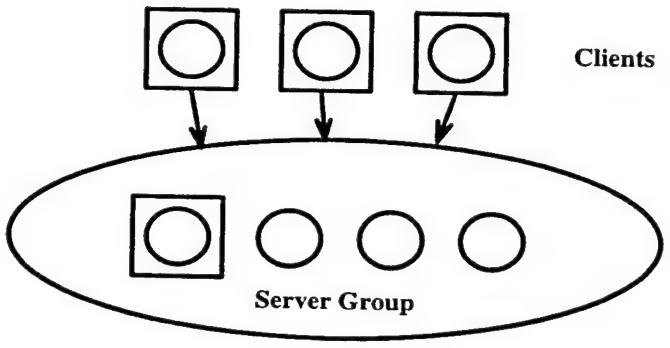



Figure 3: Executable Specification.

Executable specification can also be used for testing. For testing purposes, the user can generate a situation as shown in Figure 3. In the picture, there is a set of clients and a server group. All clients have been generated using executable specification in order to test the server code. One of the servers is also an executable specification which has bad behavior specified into it. This server is intended to test the server code of other servers in the server group. The idea is that things like "what would happen if server 1 did *this*" can be easily tested by specifying *this* in the server's specification.

## 2.3 Design for Testability

Many current protocols which are designed do not have testing code built into them. In order to be able to better test protocols which are built in the future, it is necessary that the test code be written right into the protocol so that the system can be more easily tested. If designed correctly, this test code can be left in the system and later used for system diagnostics and run-time monitoring.

Sometimes, when testing a system it is desirable to force the system into a particular execution. This may not be feasible by simply sending messages which the protocol expects. One way to accomplish this is by sending new kinds of messages which the protocol recognizes and which cause the protocol to manipulate certain data structures within the protocol address space. In this manner, the system can be orchestrated into certain states from "outside" which are not reachable by simply sending messages to the protocol participants.

For example, when testing TCP the fault injection layer might need to use and change the current sequence number. The fault injection layer may also need to put a packet on the timeout queue for resending in case of packet loss. If both of these "hooks" were written into the TCP layer code, the FI layer could generate packets and manipulate data structures in the TCP code so that the TCP layer would believe that it had generated the packets itself. What kinds of hooks protocol designers might want to include in their protocols is one of the subjects of this research.

# 3 Implementation

In the current implementation, the driver and fault injection layers are implemented as protocol layers of the *x*-Kernel [8]. *x*-Kernel was chosen because of the ease of changing the protocol stack to accommodate new layers. The fault injection layer uses Tcl [10] as the interpreter for the scripts which run when messages are pushed/popped into it. Tcl was used because the Tcl interpreter is embeddable in C code and the Tcl language is easy to write scripts in. The interpreter can also be easily modified to contain new operations which can be called from scripts. These operations are actually implemented as C code in the *x*-Kernel's address space, giving the scripts the power to modify *x*-Kernel address space and state. Common operations on messages include `msg_type`, `msg_generate`, and `msg_drop`. Script writers can also specify probability distributions. For example, a call such as `dst_normal mean var` will produce numbers with a normal distribution around *mean* with variance *var*. This can be used for actions such as dropping messages with certain probability distributions.

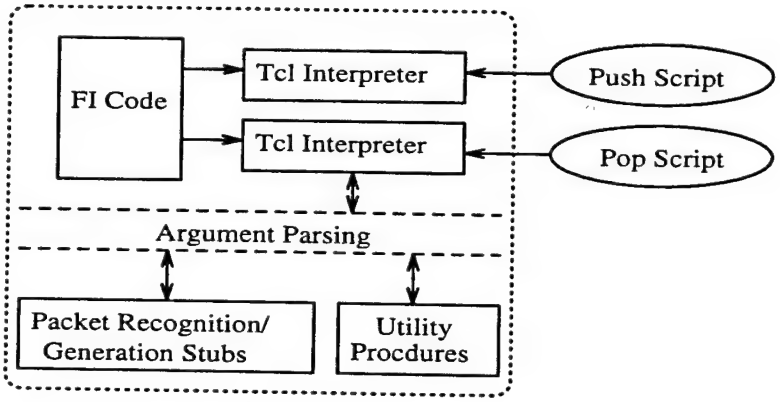



Figure 4: Fault Injection Layer Architecture.

The software architecture of the fault injection layer is shown in Figure 4. The packet recognition and generation stubs are written in C by someone who knows the packet formats of the layer which is being tested. The packet generation routine simply generates messages with the correct headers given a packet type. The packet recognition stub may be as simple as a routine which, when given a packet, returns the type of the packet. These routines are part of the Tcl interpreter and can be called from the Tcl scripts.

In order to make it easier to "plug in" different packet recognition/generation and other Tcl routines, an extra layer of abstraction is provided. This layer sits between the actual utility procedures and the Tcl interpreter. It is what is called by the scripts. These routines simply take arguments from the script and then call the appropriate procedure. They perform argument parsing and error handling the way procedures which are registered with Tcl should. For example, when figuring out the packet type, the message_type routine is called. The code for message_type takes the message handle, turns it into an actual pointer to a message, and then calls the message_type stub routine to get the type of the message. The message_type

stub routine returns the message type, which is then put into a buffer as a return value and then the bridge code returns. The script gets the buffer as the return value of message_type. When changing the packet recognition/generation routines, the user need only change the stub routine, and does not necessarily need to understand how Tcl calls work.

Each time a packet is pushed or popped in the protocol stack, the appropriate script (either push or pop) is called. The interpreter has one message handle called *cur_msg* defined. The script can do operations like:

```
msg_type cur_msg
```

and

```
msg_drop cur_msg
```

which will operate on the current message (since cur_msg was specified). `msg_generate` generates a new message with a specified type and returns a handle to the new message. This handle can be used to operate on the message. For example, the following code generates a message of type five and then pushes it down the stack.

```
set handle [msg_generate 5]
msg_push $handle
```

When a `msg_push` is done, a new thread is created in the *x*-Kernel to handle the message. It is important to note that this thread will not run until after the current thread (the one handling *cur_msg*) terminates.

# 4 Conclusion

This paper presented a framework for fault injection and testing of distributed protocols. The advantages of the proposed approach include: portability to different platforms; uniform treatment of network communication and application-level protocols; support for deterministic and probabilistic testing; and support for user-defined test scripts. Ongoing activities on this project are currently focused on three related paths: (i) development of a more elaborate tool with a graphical user interface; (ii) automatic generation of test scripts from a high-level system specification; and (iii) experimental studies of distributed protocols such as the TCP transport and the real-time channel protocols.

# References


[1] J. Arlat, Y. Crouzet, and J.-C. Laprie, "Fault injection for dependability validation of fault-tolerant computing systems.," in *Proc. Int'l Symp. on Fault-Tolerant Computing*, pp. 348–355, June 1989.

[2] D. Avresky, J. Arlat, J. Laprie, and Y. Crouzet, "Fault injection for the formal testing of fault tolerance," in *Proc. Int'l Symp. on Fault-Tolerant Computing*, pp. 345–354. IEEE, 1992.

[3] R. Chillarege and N. S. Bowen, "Understanding large system failures — a fault injection experiment," in *Proc. Int'l Symp. on Fault-Tolerant Computing*, pp. 356–363, June 1989.

[4] K. Echtle and Y. Chen, "Evaluation of deterministic fault injection for fault-tolerant protocol testing," in *Proc. Int'l Symp. on Fault-Tolerant Computing*, pp. 418–425. IEEE, 1991.

[5] K. Echtle and M. Leu, "The EFA fault injector for fault-tolerant distributed system testing," in *Workshop on Fault-Tolerant Parallel and Distributed Systems*, pp. 28–35. IEEE, 1992.

[6] G. Finelli, "Characterization of fault recovery through fault injection on ftmp," *IEEE Trans. Reliability*, vol. 36, no. 2, pp. 164–170, June 1987.

[7] S. Han, H. A. Rosenberg, and K. G. Shin, "DOCTOR: An integrateD sOftware fault injeCTOn enviRonment," Technical Report CSE-TR-192-93, The University of Michigan, December 1993.

[8] N. C. Hutchinson and L. L. Peterson, "The *x*-Kernel: An architecture for implementing network protocols," *IEEE Trans. Software Engineering*, vol. 17, no. 1, pp. 1–13, January 1991.

[9] G. Kanawati, N. Kanawati, and J. Abraham, "FERRARI: A tool for the validation of system dependability properties," in *Proc. Int'l Symp. on Fault-Tolerant Computing*, pp. 336–344. IEEE, 1992.

[10] J. K. Ousterhout, "Tcl: An embeddable command language," in *Winter USENIX Conference*, pp. 133–146, January 1990.

[11] Z. Segall et al., "Fiat – fault injection based automated testing environment," in *FTCS-18*, pp. 102–107, 1988.

[12] K. G. Shin and Y.-C. Chang, "Load sharing in distributed real-time systems with state change braodcasts," *IEEE Trans. Computers*, vol. C-38, no. 8, pp. 1124–1142, August 1989.

COMPUTER COMMAND AND CONTROL COMPANY
2300 CHESTNUT STREET, SUITE 230 • PHILADELPHIA, PA 19103
215-854-0555 FAX: 215-854-0665

# The Synthesis of Software Artifacts with Implementation: Re–creating Requirements Specifications

## 1994 Complex Systems Engineering Synthesis and Assessment Technology Workshop (CSESAW '94)

**July 19–20, 1994**
**Washington, DC**

**Dr. Judith Ahrens[†], Mr. Evan Lock, and Dr. Noah S. Prywes***
**Computer Command and Control Company**
**2300 Chestnut Street, Suite 230**
**Philadelphia, PA 19103**
**Tel: 215–854–0555, Fax: 215–854–0665**
**Email: lock@cccc.com**

[†]Also with Drexel University
*Also with University of Pennsylvania

## Abstract

There is a strong tendency in the Department of Defense Programs to re–develop new software when legacy software can be reused at lower cost, reduced development time and higher quality based on real–life experience. The decisions for re–developing software are frequently based on inadequacy, or sometimes lack, of documentation and the difficulty of understanding the legacy software. The cost of software understanding has been estimated at 50% of the cost of reengineering.

This paper describes an approach and toolset that synthesizes automatic processing of legacy code to produce a graphical explanation of the software architecture, and to generate reliable software specifications documents in accordance with DOD–STD instructions. The need for this capability is widely recognized.

This capability has been under development for the past 3 years. It consists of integrating the Software Reengineering Environment (SRE), funded by the Naval Surface Warfare Center (NSWC), and the Software Specification Assistant (SSA), funded by the Joint Logistics Commanders–Joint Policy Coordinating Group on Computer Resource Management (JLC–CRM). The development is nearing completion and a demonstration project is planned.

## 1. MOTIVATION

At a recent workshop on Reengineering (Fourth Systems Reengineering Technology Workshop, February 8–11, 1994, Monterey, CA), several speakers reported that understanding legacy software accounts for 50% of the cost and time of reengineering. At this high cost, Program Managers tend to write off all or parts of the legacy software and develop new system modules or entirely new systems. Thus, there has been a widely recognized need for automating software understanding.

Another widely recognized problem is the frequent unreliability, incompleteness and sometimes total lack of software documentation. Software documentation is produced in many software development projects as the last step and tends to be short–changed. There has been no effective procedure to determine the quality of submitted documentation. The inadequacy of documentation has also prevented verification that the software provides the capabilities established in planning, specifications and contracting documents.

Programs across DOD need to be able to:

i. Check conformance with Software Specifications in periodic Contractor reviews and upon delivery of a new system,

ii. Update obsolete specifications,

iii. Create specifications for undocumented software,

iv. Understand existing software architecture (for reuse).

The proposed capabilities will have a wide ranging impact on:

i. Reducing maintenance costs by graphically explaining the architecture and operation of the software.

ii. Increasing system life through adding new builds incrementally based on explaining graphically the architecture and operation of the software.

iii. Facilitating modernization through exposing the steps necessary to execute the system in a modern distributed computer communications environment.

iv. Improving quality and usefulness of systems through facilitating verification of a system in reviews that assure conformance with the requirements, specification and contract for the system.

The next section describes the two toolsets (and their interface) that are the basis for providing the above capabilities. The third section describes how the two tools are used together to define an approach to re–create documentation. Section 4 reviews status and plans.

## 2. TECHNOLOGY

Two automatic tools, used in the automation of software understanding and documentation, are shown in Figure 1. They are:

- The Software Reengineering Environment (SRE) [SRE]: It has been developed under the sponsorship of the Naval Surface Warfare Center (NSWC). It incorporates software translation to Ada (from CMS–2 and in the future from FORTRAN) and the abstracting of Ada code to re–create graphically the architecture and the data and control flow.
- The Software Specification Assistant (SSA) [SSA]: It provides tools (COTS) for searching historical documents and the editing and formatting necessary for creating and updating software specifications.

Each of these environments are described further below. These descriptions provide the basis for explaining how the capabilities are synthesized as part of a cohesive approach for document re–creation.

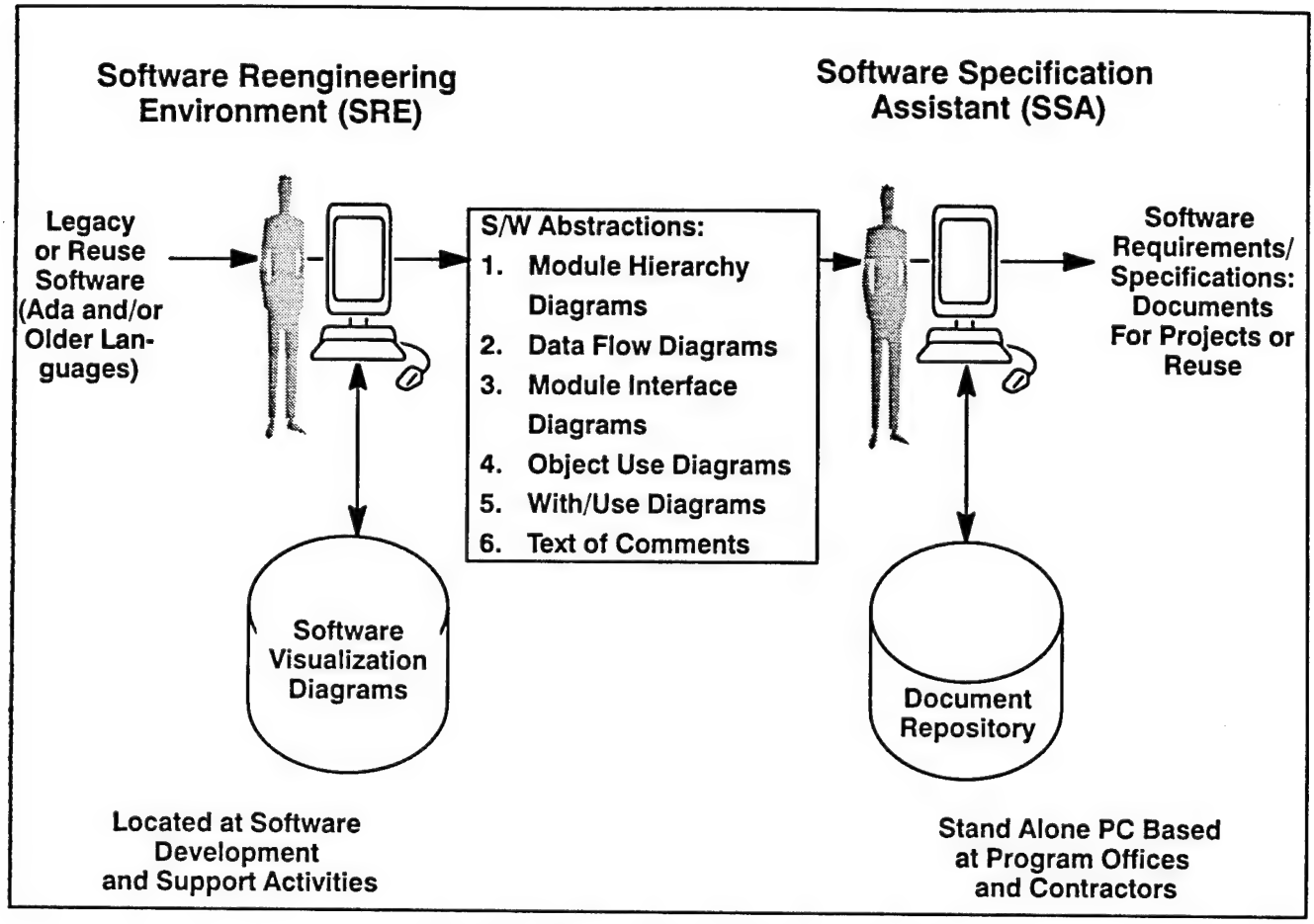



**Figure 1: Process of Re–creating Software Specifications.**

## 2.1 Software Reengineering Environment (SRE)

SRE incorporates the technologies of software translation, visualization, and understanding. SRE's architecture and capabilities are shown in Figure 2. The SRE consists of two phases, *Software Restructuring* and *Software Understanding.*

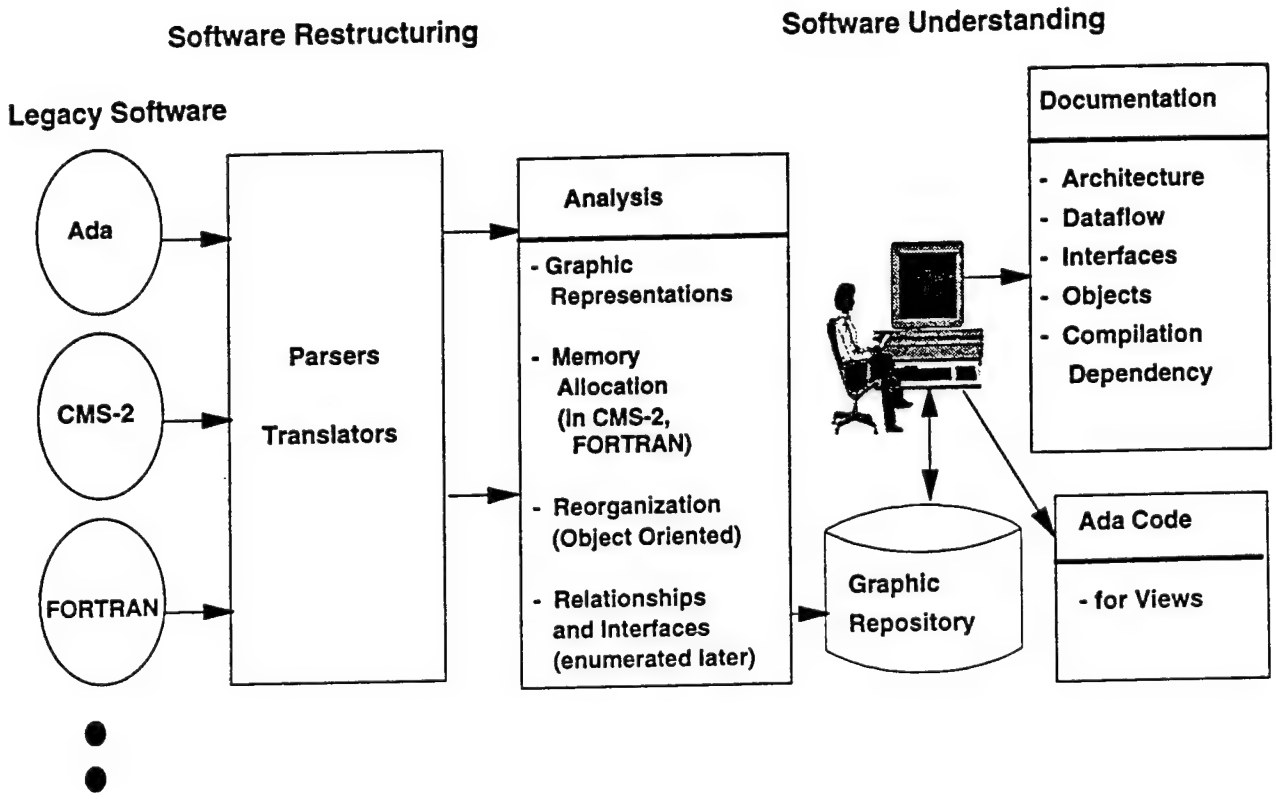



**Figure 2: Architecture and Capabilities of the Software Reengineering Environment (SRE)**

Software Restructuring parses and translates CMS–2, Ada and, in the future, FORTRAN code, statement by statement, into Entity–Relation–Attribute (ERA) diagrams of pseudo–Ada [PRYW]. This diagramming scheme is called Elementary Statement Language (ESL) for Ada. Next, the ESL–Ada is transformed repeatedly to obtain an Ada programming paradigm in a series of passes that achieves 100% translation to Ada. Each pass translates different aspects of the programming paradigm of the source language into the Ada programming paradigm (e.g., separating object specifications from bodies). During the translation process, a number of sets of relations among program statements are generated. The statements form nodes, and the relations form edges, in the ESL–Ada graphic diagrams.

Software Restructuring partitions the software into multi–level hierarchical software components. Software Abstraction Documents (shown in Table 1) are then generated. They describe hierarchically the architecture of these components from different perspectives. Component diagrams are stored in a graphic repository.

| Abstraction Document | S/SS System/Segment | S/SDD System/ Segment | SRS CSCI | IRS CSCI |
|---|---|---|---|---|
| Hierarchy Diagram | Par. 3.1, 3.2.3 System Architecture Diagram | Par. 4 System Architecture Diagram | Par. 3.1 CSCI External Interface Diagram | Par. 3.1. CSCI Internal Interface Diagram |
| Flow Diagram | | | Par. 3.3 CSCI Internal Interface Diagrams | |
| Interface Table | | | Par. 3.3 CSCI Internal Interface Diagrams | Par. 3.x.1 Data Element Table |
| Context Diagram | For Ada Compilation | | | |
| Object/Use Diagram | For Object Orientation | | | |
| Comments Text | For Capabilities | | | |

**Table 1: Mapping Software Abstraction Documents into Software Specifications.**

Software Understanding (SU) consists of query and retrieval of graphic diagrams that illustrate the software from various perspectives. A graphic query language is provided for ad hoc browsing of the Software Abstraction Documents in the graphic repository. These graphs show relations between high or low level hierarchical components. This facilitates understanding of the software's architecture as well as its detailed code [PRWY3]. Facilities are being developed to make changes to the program, for debugging or program restructuring, via the graphics used for visualization. Software visualization overcomes the essential invisibility (i.e. non–physical quality) of software by representing graphically the program structure, control flow, and data. An abstract, graphical representation can facilitate a software engineer's visual perception and cognitive understanding of complex software during debugging, monitoring, and especially, program restructuring. In this way, maintenance can be performed on the reverse engineered design and/or transformed old code.

## 2.2 Software Specification Assistant (SSA)

SSA [SSA] was designed for project technical management. It is an integrated set of information repositories and tools for software specification of critical mission systems. It instructs and informs novice to expert staff in specifying, updating and evaluating DoD–STD data item descriptions (DIDs), including the System/Segment Specification (SSS), System/Segment Design Document (S/S, Software Requirement Specification (SRS) and Interface Requirements Specification (IRS) of DoD–STD–2167A (and similar documents on its planned successor DoD–STD–SDD) [2167A].

SSA maximizes the effectiveness of supervisory staff who are experts in the preparation of requirements specifications, and provides an automated mechanism for novices to upgrade their skills. SSA thus provides two modes of operation. In supervisory mode, using the Status Manager subsystem, the supervisor structures the requirements specification tasks and monitors progress. In activity mode, the Step–By–Step subsystem, guides a novice specifications analyst through the required work processes. SSA thus embodies much of the knowledge found in supervisory staff, enabling an organization to make efficient use of this scarce organizational resource.

SSA is composed of four customized subsystems. The function of the subsystems is as follows:

- Documentation Manager is used to create catalogs of application and reference documents in databases.
- Assignment Manager is used by a manager or supervisor to enter the work plan for staff who compose or update documents.
- Step–By–Step guides users engaged in searching documents and composing/updating Requirements and Data Items.
- Evaluate provides feedback on the completeness of the specification coverage [ARTH].

SSA also integrates the following commercial off–the–shelf software (COTS): document loading and publishing (e.g., Interleaf, MS–WORD, or Word Perfect), Search (Zyindex), CASE (depends on use by the Program Office).

The Assignment Manager subsystem enables a supervisor to create a documentation plan and assign subordinates. The process is accomplished by selecting the appropriate function from the Status Manager pulldown menu. For each project, tasks are allocated, organized and controlled through a hierarchy of three lists: Things to Do, Target Documents, and Target Document Paragraphs.

In the **Things To Do List** the supervisor enters the tasks that need to be accomplished. Examples include: work on entries from an operational requirements document's table of contents, work on items from a functional decomposition, or work on items requiring specification. For each task, the supervisor references a previously loaded document (or equivalent) that expands on the item in the Things To Do List. For example, clicking on the Things to Do list reveals a definition of its entries.

The **Target Documents List** contains the names of specification documents to be created or updated. The **Target Document Paragraph** contains the Paragraphs to be created.

At each of the three list levels, the user and the supervisor can record relevant instructions or status information such as priority of the item, problems encountered in completing the item, or sources of information used to complete the item.

After organizing the work needed to complete a plan, the supervisor assigns the work to subordinates. The subordinate will use the Step–By–Step subsystem.

The **Step–by–Step** Subsystem guides the user through the process of preparing requirements specifications. Step–by–step is an iterative process. Once a task and associated target documents are selected, the user iterates, in various combinations (even during different sessions), to search for application information and assistance, to compose data items, and to record a trace, until the selected task is completed. Then, the user will select another task from the list of Things to Do and repeat the process.

## 3. Approach: Interfacing SRE and SSA

The previous sections provided background on SSA and SRE. With these two capabilities available, the next question to answer is – what are the necessary abstractions that the SSA user needs to recreate specifications and can SRE produce these abstractions? The answer is that there are six basic types of abstraction/information (see Figure 1) that comprise the interface between these two tools [PRYW2].

The diagrams are created by traversing the repository for nodes with the following relations:

1. **Hierarchy Relations**: The entire repository is envisaged structured as an upside–down tree–like hierarchy. The root unit of the tree is called a *System.* Its immediate descendants are called *Segments*. Segments can have as descendants Segments or *Computer Software Configuration Items* (CSCI). CSCIs can be object declarations, database declarations or major executable code units. CSCIs can have multiple levels of descendents called *Software Units* (SU). Software specifications document requirements/capabilities associated with each System, Segment or CSCI module in the repository.
2. **Architecture Unit Relations**: These relations are specified for each architecture unit. The interfaces are through data, transferred to or from the module or through I/O or through references.
3. **Data–Flow Relations**: These relations provide information on units that participate in a Data Flow diagram of a process accomplished by modules. The data flow relations are implemented in the programs by I/O, procedure calls or message passing.
4. **Type–Instantiation Relations**: These relations relate units that contain type (and generic) declarations with those where these declarations are used.
5. **With/Use Relations**: These relations relate units that are users of other units in a library of programs.
6. **Text of Comments** – These are related to modules through keywords.

The software abstraction process combines the above relations to produce Application Abstraction Documents (AAD). Each document is named, identifies the software being documented, specifies what kind of document it is, and specifies what are its relations to other documents. Units are either Systems, Segments, Computer Software Configuration Items (CSCI), or Software Units (SU).

The information collected during the software abstraction process is presented in six kinds of documents. All the documents focus on the high level units (systems, segments, CSCIs) of the software being abstracted. Two kinds of documents deal with the relations that exist between units:

1. **Module Hierarchy Diagrams** specify the part–of relation
2. **Context Diagrams** specify the visibility relation

**Object Use Diagrams** specify the subclass and instantiation relations that exist between units, and between types and data structures. Three additional kinds of diagrams describe individual units:

1. **Unit Structure Diagrams** specify the internal structure of a unit and its internal and external interactions.
2. **Interface Tables** describe in tabular form the interactions between a unit and its environment.
3. **Comment Sections** contain the comments associated to units.

No application abstraction document at present provides information on the dynamic behavior of the software being abstracted. In particular, no state diagram, event diagram, or timing diagram is

produced. At present, timing information must be obtained through existing documentation, simulation and/or instrumentation of the source code.

The above collection of graphic views is prepared by the SRE user. The diagrams are exported from SRE, catalogued by SSA's Document Manager, and loaded into SSA's search system. This can be accomplished electronically or through scanning. These diagrams and tables can be searched and portions retrieved to satisfy user interests. The SSA user progressively searches these diagrams along with prior requirements documents or other related application information to attribute capabilities and non–functional requirements to the diagrams. The diagrams can be cut and pasted directly into the appropriate sections of the requirements specification as shown in Table 1. This searching of the diagrams can also serve as the basis for exploring commonalities and variabilities of requirements for domain/application engineering.

## 4. Status and Plans

The implementation of the converged SRE/SSA system will be completed during the summer of 1994. The plan is to follow this with a demonstration project to evaluate the system's usefulness and effectiveness. The demonstration will consist of processing existing legacy code and producing the necessary understanding and documentation. The demonstration will also compare existing software documents with those produced by the automatic system from the code. The steps in the project include:

**Step 1:** Selection of a software system to be used in the demonstration project with a participating DOD agency. The software system will have the following characteristics:

i. Existing software specification for later comparison with the automatically produced documentation.

ii. Existing Ada code of significant size (e.g., up to 1000,000 lines of code).

This step will involve interviewing the DOD agency's programs and staff. The selected software will have to be available for automatic processing by CCCC.

**Step 2:** Process the selected code in the SRE. Produce the software abstraction reports discussed in Section 3.

**Step 3:** Transfer the software abstraction reports from SRE to SSA.

**Step 4:** Load the existing software specifications into SSA.

**Step 5:** Produce software specifications for the selected software.

**Step 6:** Compare the new and old specifications and produce a list of differences.

## 5. Bibliography


[2167A] DOD–STD–2167A: Defense System Software Development, September 1988.

[AHR] Ahrens, Judith, N. Prywes, and E. Lock. 4th Systems Reengineering Technology Workshop, "Maintenance Process Reengineering: Toward a New Generation of CASE Technology," Monterey, CA, February 8–10, 1994.

[ARTH] Arthur, J. D., R. E. Nance, "Developing an Automated Procedure for Evaluating Software Development Methodologies and Associated Products," Technical Report SRC–87–007, System Research Center, Virginia Polytechnic Institute, 1987.

[SSA] *Software Specification Assistant User's Guide: Status Manager and Step–by–Step Guide, Document Manager Guide, Evaluation Subsection Guide and Installation Guide*, delivered to the Joint Logistics Commander Computer Resource Management Sub–Group and the Office of Naval Research by Computer Command and Control Company as part of contract #N00014–91–C–0160, December 1992.

[LOCK] Lock, E. and N. Prywes, Tri–Ada '92 Conference, "Requirements on Ada Reengineering Technology from Past, Present and Future Systems," Orlando, FL, November 16–20, 1992.

[PRYW] Prywes, Noah, G. Ingargiola, I. Lee, and M. Lee, 4th Systems Reengineering Technology Workshop, "Reengineering Concurrent Software to Ada," Monterey, CA, February 8–10, 1994.

[PRYW2] Prywes, N., Ingargiola, G. and Ahrens, J. "Automatic Reverse Engineering of Software to Confirm/Update Requirements Specification," Computer Command and Control Company, Contract No. N00014–92–C–0242, Philadelphia, PA, 19103, June 1993.

[PRYW3] Prywes, N., Lee, I. "Integration of Software Specification, Reuse and Reengineering," Computer Command and Control Company, Contract No. N60921–92–C–0194, Philadelphia, PA, 19103, June 1993.

[SRE] *Software Re–engineering Environment (SRE) Demonstration Guide*, Version 5.0, Computer Command and Control Company, Prepared Under Contract N60921–92–C–0916, White Oak Lab., Silver Spring, MD, May 6, 1994.

# Software Tools for Formal Specification and Verification of Distributed Real-Time Systems: A Progress Report

J. Choi[†], I. Lee[†], J. Kim[‡], and E. Lock[‡] *

[†] University of Pennsylvania
Department of Computer and Information Science
Philadelphia, PA 19104

[‡] Computer Command and Control Company
2300 Chestnut Street, Suite 230
Philadelphia, PA 19103

1994 Complex Systems Engineering
Synthesis and Assessment
Technology Workshop (CSESAW '94)
July 19-20, 1994
Washington, DC

*Supported by a grant, N00014-94-C-0081, from ONR.

# 1 Introduction

Real-time systems are widely used in control systems (manufacturing systems, robotics), monitoring systems (patient monitoring, air traffic control), communication systems and weapons systems. The correctness of a real-time system depends not only on how concurrent processes interact, but also on the time at which these interactions occur. Since the cost associated with an "incorrect" operation of these systems can be extremely high, a rigorous way of specifying, analyzing and evaluating various design alternatives is demanded.

We plan to build a toolset environment based on formal models for specification and verification of large distributed real-time systems, and to evaluate the effectiveness of the environment experimentally. The environment is based on a hierarchical specification paradigm and resource-based computation models.

This research project is supported by a Small Business Innovation Research (SBIR) Program from the Office of Naval Research (ONR). As the first part of this project, we have examined some of the state-of-the-art specification methods and their implementation. They are briefly summarized in Section 2. Section 3 describes the on-going work. Concluding remarks are presented in Section 4.

# 2 Results from the Current Investigation

We have examined the following three formal specification languages: 1) Modechart, 2) Algebra of Communicating Shared Resources (ACSR), and 3) ModularChart. Modechart is a graphical specification language. A subset of Modechart has been implemented as MCtool[6] by the University of Texas at Austin and NRL. MCtool has a nice, user-friendly, graphical user-interface, a graphical editor, and various analysis and display capabilities. ACSR, developed at the University of Pennsylvania, is a real-time process algebra with the notions of time, resources and priorities. Similar to other process algebras, ACSR has well-defined semantics, the notion of equivalence based on bisimulation and a sound and complete set of laws. It, however, lacks a graphical interface. A new graphical specification language called ModularChart has been carefully designed as a graphical front-end to ACSR. However, it turns out to be too general and expressive.

The graphical specification language can be implemented either by using a meta CASE tool to accelerate building a completely new tool or by augmenting an existing graphical system such as MCtool. We have studied the possibility of using a meta CASE tool (Meta-DoME). We are now investigating the possibility of using Modechart as a graphical front-end for ACSR. We need to perform a number of trade-off analyses before we decide the implementation method.

## 2.1 ModularChart

A number of elegant techniques have been developed for graphical specification languages. Among them are Statechart and Modechart. One interesting observation is that transitions in these formalisms are not level restricted: any transition may leave one mode and enter another at any level. Not restricting a transition may increase expressive power; however, such unrestricted transitions can make a specification non-hierarchical.

A new graphical specification language has been examined for this project: ModularChart which is strongly motivated by Modechart. ModularChart is a strictly hierarchical, modular, real-time specification language. A transition must be defined between two sibling modes so that a transition leaves from one mode to another at the same level. This kind of restriction is similar to the scoping rules of existing high-level programming languages. In addition, ModularChart has clocks and various types of variables, as well resources and priorities.

ModularProcess is a textual specification language. The basic constructors of the language are processes and transitions. In addition to these constructors, ModularProcess has variable declarations and assignment statements as well as clocks. The language is simple but very expressive since many important features (such as deadline, interrupt, scope and timeout) of real-time system description languages can be encoded using ModularProcess constructors. ModularProcess and ModularChart are identical in terms of semantics and there is a natural isomorphism between the specification languages. Having the same semantics, these two specification languages as a whole are good candidates for specification of large, real-time systems. The combination of languages support easy-to-see visual hierarchical specifications as well as convenient textual specifications.

Although ModularChart was designed to be a front-end to ACSR, it became too general and expressive. It would be more effective for us to use Modechart as a graphical specification language. Section 3 describes what has been done and needs to be done to integrate Modechart and ACSR.

## 2.2 Modechart

Modechart[4] is a graphical specification language developed to provide a compact and structured way to represent real-time systems. Figure 1 shows an example of Modechart specification. Modechart is a variant of the Statechart language. The motivation of Modechart is given as follows [3]:

- Statechart is too liberal in permitting the forms of predicates to appear in the conditions for enabling state transitions. As a result, it is difficult to prevent anomalies in defining a semantics for Statechart.
- Statechart does not provide adequate treatment of stringent timing constraints.

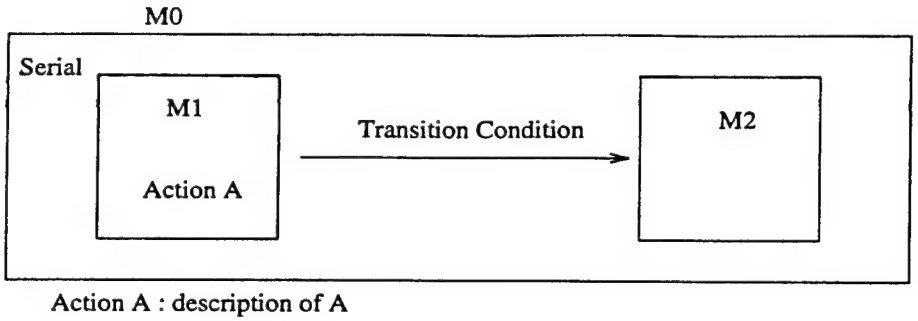



Figure 1: An example of Modechart

One known limitation of Modechart is that its semantics are ambiguous or unclear as mentioned in [1, 7]. Because of the semantical ambiguity, the current Modechart implementation does not support the full features of Modechart.

## 2.3 ACSR

ACSR is a real-time process algebra that incorporates the notions of communication, concurrency, resources, and priorities into a single formalism. One of the important concepts in ACSR is shared resources. The detailed description and semantics of ACSR can be found in [5].

The syntax of ACSR processes containing actions is as follows:

$$P ::= \text{NIL} \mid a.P \mid A : P \mid P_1 + P_2 \mid [P]_I \mid P_1 \| P_2 \mid rec\ X.P \mid P \triangle_t Q \mid P \uparrow Q \mid X$$

NIL is a process that executes no action (i.e., it is deadlocked). There are two prefix operators. $A : P$ executes a resource-consuming action $A$, consumes one time unit, and proceeds to the process $P$. $a.P$ executes an instantaneous action $a$, and proceeds to the process $P$. The Choice operator $P_1 + P_2$ represents nondeterminism – either of the processes may be chosen to execute, subject to the resource limitations of the environment. The operator $P_1 \| P_2$ is the concurrent execution of $P_1$ and $P_2$. The Close operator, $[P]_I$, produces a process $P$ that monopolizes the resources in $I \subseteq \mathcal{R}$. The Timeout operator, $P \triangle_t Q$ monitors the time passing of the process $P$ and when time out occurs, the control is transferred to the process $Q$. The Interrupt operator $P \uparrow Q$ denotes that the process $P$ can be interrupted during its execution and when interrupt occurs the process $P$ stops and the process $Q$ must be executed. The process $rec\ X.P$ denotes standard recursion, allowing the specification of infinite behavior.

ACSR supports the notion of equivalence called prioritized strong bisimulation. There is a sound and complete set of laws for finite state processes. We also have developed a set of simple tools for ACSR that can be used to syntax check ACSR terms, interactively execute terms, rewrite terms, and test bisimulation equivalence of two terms.

## 2.4 MetaDoME

MetaDoME (Meta-Domain Modeling Environment) is a graphical tool for generating tools of graphical modeling [2]. It has been developed by Honeywell Systems and Research Center. Their goal is "to develop an open, extensible graph editing framework into which domain-specific engineering knowledge and tools can be integrated to automate the software design process.[2]" The collection of the generated graph editing tools is referred as the DoME system. DoME is implemented on GrapE which is a set of Smalltalk-80 classes and methods for graph editing. DoME supports the following features which are rarely found in "conventional" graph editing tools:

- Nodes can be complex entities, each of which can visually and semantically contain an entire graph.
- Any graph object may have multiple subgraphs.
- Changes can be automatically propagated between connected graphs.
- Each graph can have a user-defined hierarchical set of views.
- Many sources of inconsistency can be eliminated via high degree of connectivity.

MetaDoME can provide a rapid prototyping process that involves iteratively specifying objects of a target graphical tool. The process is performed via a Visual Programming methodology and based on the Object-Oriented design method. Once a specification of the target tool is composed, MetaDoME automatically generates Smalltalk code for the tool based on the specification. The MetaDoME environment and the generated graphical tool offer a high degree of portability in the sense that it can be installed under various operating systems such as Unix, VMS, Windows, Macintosh, etc.

# 3 Current Research Direction

Based on the results from the previous works, we are currently investigating how to integrate the Modechart and a process algebra, ACSR, into a graphical specification language. As the first step, we are currently studying how to translate ACSR to Modechart and how to translate Modechart to ACSR. The goal of this investigation is to combine ACSR and MCtool, which is based on Modechart. The well-thought-out integration would result in several advantages as follows:

- Hierarchical graphical specification language with a graphical user interface.
- Several possible verification methods based on ACSR rewrite rules, ACSR equivalence testing, simulation provided in MCtool, and model checking of Modechart.
- Well-defined Semantics (Semantics based on process algebra are well-defined and clear).

- Support two basic communication methods: synchronization and broadcasting.
- Support the resource and priority concepts.

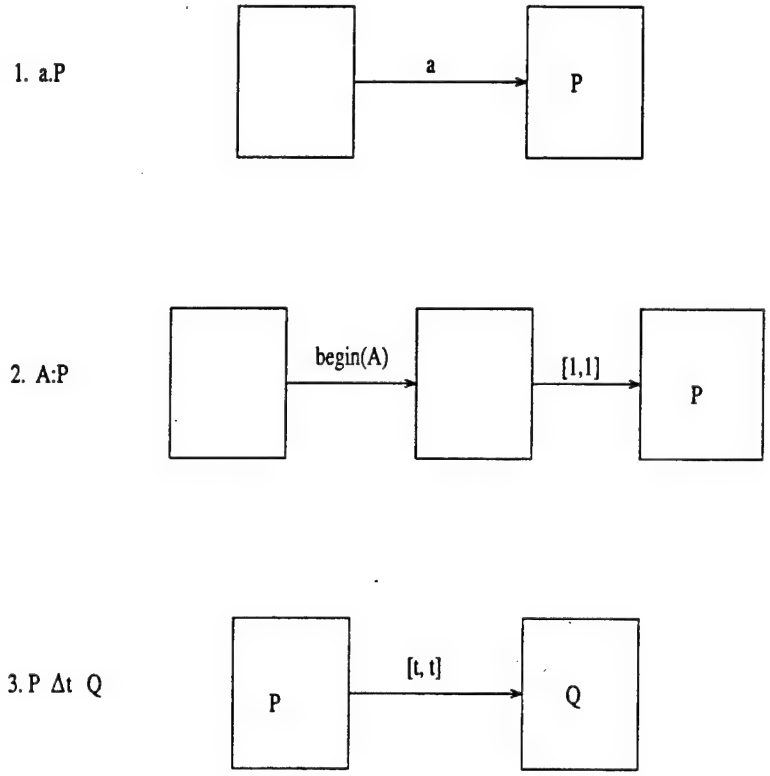



Figure 2: An example of translation of ACSR to Modechart

## 3.1 Translation from ACSR to Modechart

Since ACSR can be regarded as a restricted form of Modechart in terms of structure, translation from an ACSR term to a Modechart specification seems straightforward as seen in Figure 2 except for synchronous communication, priorities, resources and resource closure. We are working on how to extend Modechart to support them. This study is to determine how MCtool can be used as a graphical front-end to ACSR based tools.

Figure 2 illustrates how ACSR terms can be translated into Modecahrt. It is clear that two prefix operators (. and :) should be translated into serial modes, since the semantics of prefix operators are identical to serial modes. Similarly, timeout $t$ in ACSR can be translated into serial mode with timing information $[t, t]$.

## 3.2 Translation from Modechart to ACSR

It seems that a subset of Modechart can be translated into ACSR. Such a subset has the semantics of ACSR, hence, no ambiguity is possible. In this case, a real-time system can be

specified using MCtool and the correctness of the system can be verified using ACSR based analysis tools.

There is one major problem with this approach because ACSR is based on synchronous communication, whereas Modechart is based on broadcasting. So, we are currently investigating how to add broadcasting to ACSR. In ACSR, we introduce two symbols, ?? and !! for broadcasting: $a??$ denotes for receiving an event through the broadcasting channel $a$, and $a!!$ for sending an event. A part of the proposed operational semantics for $a??$ and $a!!$ are as follows:

$$\frac{P \xrightarrow{a!!} P', Q \not\xrightarrow{a??}}{P \| Q \xrightarrow{a!!} P' \| Q} \qquad \frac{P \xrightarrow{a!!} P', Q \xrightarrow{a??} Q'}{P \| Q \xrightarrow{a!!} P' \| Q'}$$

$$\frac{}{a??.P \xrightarrow{\emptyset} a??.P} \qquad \frac{Q \xrightarrow{a??} Q', P \xrightarrow{a??} P'}{Q \| P \xrightarrow{a??} Q' \| P'}$$

The above operational semantics are basic concepts of broadcasting communication which are based on the following two facts:

- Receiver should wait indefinitely until sender sends a message.
- Sender should proceed to next precess immediately after sending a message without checking the existence of receivers.

In order to model broadcasting communication using ACSR, $a!!$ should not be restricted, but $a??$ must be always restricted.

# 4 Conclusion

A formal, hierarchical specification language facilitates the design of a reliable, large-scale, complex system. The formal aspect of the language reduces ambiguities in the specification and supports mathematical analysis techniques. Its hierarchical aspect, on the other hand, allows designers to control the amount of detail to expose during different phases of the design. Hiding details that are irrelevant to a particular phase of the design helps to focus on that design phase. An abstract specification, for example, can be analyzed for functional and timing properties before it is refined to include details about how the functions are internally structured. Another advantage of a hierarchical language is that it can be displayed graphically. A graphical specification language helps to visualize the structure of a complex system. By visualizing various components of a complex system one can understand easily how the system's components interact and avoid various mistakes in the system design stage.

In order to develop a specification language satisfying the above requirements, we have developed a new hierarchical specification language, called ModularChart for real-time systems. We have also examined the existing Modechart language and system. At this moment, the appropriateness of ModularChart is not clear since there are a few research issues regarding the semantics of ModularChart that have not been resolved. Based on our experience in

designing ModularChart and our examination of Modechart and MCtool, we are currently investigating how to integrate Modechart and ACSR so that we can reuse as much MCtool as possible.

The graphical specification language can be implemented either by extending the current MCtool or by using a tool generation tool such as MetaDoME. Before deciding on a proper implementation method, we need to perform a number of trade-off analyses.

# References


[1] J. Choi and I. Lee, *Comments on Modechart Semantics*, Internal memo, 1994.

[2] J. Krueger, E. Engstrom, J. Ward, MetaDoME: A Rapid Prototyping Tool Supporting Graphical Modeling Tool Development, Honeywell Technical Report: CS-R93-021, November 29 1993.

[3] F. Jahanian, R. Lee, and A.K. Mok, Semantics for Modechart in Real Time Logic. in *21th HICSS* Jan. 1988.

[4] F. Jahanian and A. K. Mok, Modechart: a Specification Language for Real-Time Systems.

[5] I. Lee, P. Brémond-Grégoire, and R. Gerber. A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems., *IEEE Proceedings*, Jan. 1994.

[6] A. Rose, M.A. Pérez, and P. Clements, *Modechart Toolset User's Guide*, NRL, 1994.

[7] D. Stuart and P. Clements, Clairvoyance, Capricious timing Faults, Causality, and Real-Time Specification, *IEEE RTSS* 1991.

# AN ENVIRONMENT TO SUPPORT DESIGN STRUCTURING

## 1994 Complex Systems Engineering Synthesis and Assessment Technology Workshop (CSESAW '94)

**July 19–20, 1994**
**Washington, DC**

**Dr. Jee-In Kim and Mr. Evan Lock**
**Computer Command and Control Company**
**2300 Chestnut Street, Suite 230**
**Philadelphia, PA 19103**
**Tel: 215–854–0555, Fax: 215–854–0665**
**Email: kim@cccc.com**

## Abstract

This paper presents a framework for performing Design Structuring (DS) techniques. DS aims to produce "optimal" (or "near optimal") system designs. The framework supports repeated design and evaluation iterations performed by systems engineers. To develop large–scale, complex systems and be able to evaluate the trade off amongst numerous alternatives, an integrated set of automated tools for DS is necessary. A DS environment called Design Structuring Assistant (DSA) has been developed. It facilitates DS activities by providing tools for specifying DS objectives, guiding DS activities, retrieving reference materials and information about DS, performing DS techniques, managing DS activities and interfacing with system design evaluation tools. This paper presents an overview of DSA and an example of using DSA.

## 1. Introduction

This paper describes a framework for performing Design Structuring (DS). An environment called the Design Structuring Assistant (DSA) is implemented based on this framework. A systems engineer uses DSA in the process of creating, updating, and evaluating "optimal" (or "near optimal") system designs with respect to a set of design objective. DSA provides the system engineer with guidance, access to a knowledgebase of systems engineering and design structuring technique information and the ability to execute (and manage) a set of design structuring techniques.

The intent is to facilitate the evolution of system designs, particularly with regard to how well they satisfy non–functional requirements, such as performance, cost, security, etc. Expression of these non–functional requirements has been described in a technical report on System Design Factors (SDF) [NgHo]. This paper will refer to these requirements as SDFs. SDFs form the basis for characterizing the design structuring objective. The techniques provided relate to maximizing or minimizing a taxonomy of SDFs as defined in [NgHo].

Research and development of DSA is sponsored by the Navy's Engineering of Complex Systems (ECS) Technology Block Program. The ECS program is managed by the Naval Surface Warfare Center (NSWC) and is comprised of several distinct projects. The Systems Design Synthesis (SDS) project contains a task which focuses on Design Structuring and Allocation Optimization (DSAO). The DSAO task has responsibility for building an environment called DeStinAtiOn. DSA is a subsystem within DeStinAtiOn [CCCC94, KiLo].

This paper is organized as follows: The framework of DSA is described in Section 2. Section 3 presents sample DS activities using DSA. The status of implementing DSA and the research plan is discussed in Section 4.

## 2. Design Structuring Framework

### 2.1. Overview

The overall view of an environment which supports Design Structuring is shown in Figure 1. There are several key subsystems:

1. **Objective Formulation:** A system engineer (SE) provides the context and the objectives of Design Structuring. The objectives will be used in the "Guide", "Perform", and "Manage" subsystems. The SE may need information provided by the "Reference" subsystem in specifying his objectives. The SDF taxonomy is used to characterize non–functional objectives.
2. **Guidance:** DSA provides "guidance" to the SE. To formulate the correct guidance, DSA needs inputs such as the SE's objectives and the results of the previous DS experience and research. The SE may follow the guidance or override it when making design decisions.
3. **Perform DS Techniques:** A selected DS technique is performed to issue recommendations so that the SE can produce or modify a system design. The SE may modify the system design based on the recommendations.
4. **Evaluation:** The produced system design is evaluated with respect to the SE's design objectives. (Note that it is expected that DSA will interface with an evaluation subsystem and that an evaluation subsystem will not be developed as part of the DSA effort.)
5. **Reference:** To facilitate DS activities such as formulating DS objectives, guidance, and design evaluation, the "Reference" subsystem serves as a repository of guidance information.
6. **Management:** The "Management" subsystem maintains the current status of DS activities. It also shows the DS activities that have been performed and possible next steps.

To complete the tasks of the subsystems, DSA needs a set of information tables. Diagram 1 represents the tables as squares and details information flow.

### 2.2. Formulation of DS Problem.

DSA requires that the SE specify design objectives before any Design Structuring activity begins. The SE states the objectives textually and in terms of a description, keywords, and SDFs.

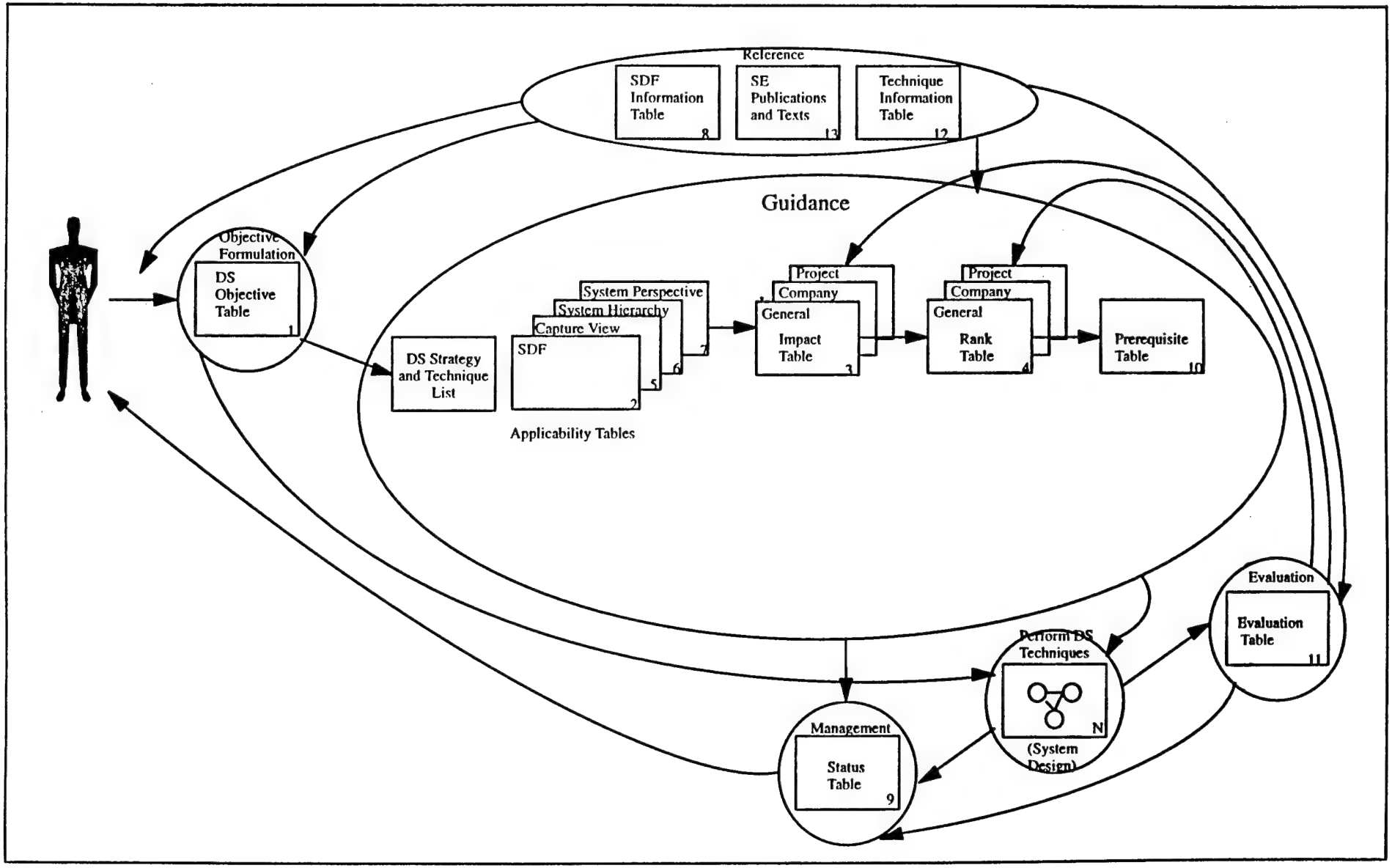



**Diagram 1: Data Flow of DSA.**

## 2.3. Guidance for Deciding DS Techniques.

There are many information tables that help SEs in selecting a DS technique:

"DS Technique Applicability Table"
"Impact Table"
"Rank Table"
"View Applicability Table"
"System Hierarchy Scope Table"
"SDF Information Table"
"DS Technique Information Table"

They will be used as "guidance" for system engineers to select the "best" technique.

For example, the "Impact Table" is defined in terms of what techniques have impact (positive, negative, no–effect, or not–available of General, Company, and Project) on the SDF hierarchy [NgHo]. Table values are based on research and experience and may range from general direction (+,–,0) to textual references. For instance, a project may say that Booch's object–oriented software design method does not address an SDF such as safety. The Company or general experience may differ.

The ranks of techniques by SDF (described in the "Rank Table") denote order of preference of the recommended techniques. The order depends on General, Company and Project research and experience. The rank is based on either qualitative or quantitative metrics. If the metric is qualitative, we first define a "quality value" and define "characteristics" based on the quality value range. The ranks can be updated after performing trade–off analysis.

## 2.4. Evaluation of System Design

Once a system design is obtained, the SE performs trade–off analysis. The results from the analysis and DS decisions are recorded in the "Analysis Log Table." Based on the results from performing DS and evaluation, DS Technique Applicability Table, Impact Table, Rank Table and View Applicability Table are updated.

## 2.5. Automatic and Interactive Modes

DSA has two operation modes:

a. The "automatic" mode is for an inexperienced SE. The SE provides "minimal" information about system design objectives. DSA then suggests a set of techniques to be applied by searching its "knowledge base" with the provided objectives. He can follow the recommendation or "override" it to take another action. At least, when he overrides the recommendation, he should specify his "rationale".

b. The "interactive" mode assumes that the system engineer is an expert and is experienced in performing Design Structuring. Therefore, the experienced system engineer dictates the steps of Design Structuring as desired, based on the information provided by DSA.

## 3. Example

The system engineer would like to synthesize and trade–off multiple designs that optimize single criteria objectives and eventually for multiple criteria objectives. The objective can be represented in terms of values for System Design Factors (i.e., non–functional requirements) such as performance, dependability, etc. For consistency, the SE would execute one or more DS techniques in order to accomplish his design objective.

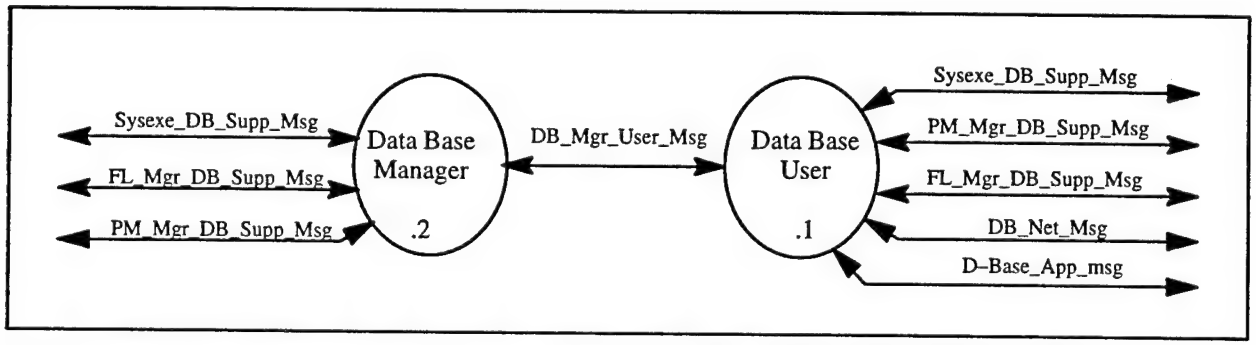



**Diagram 2: Data Base Management System of Passive Sonar System [HoKH].**

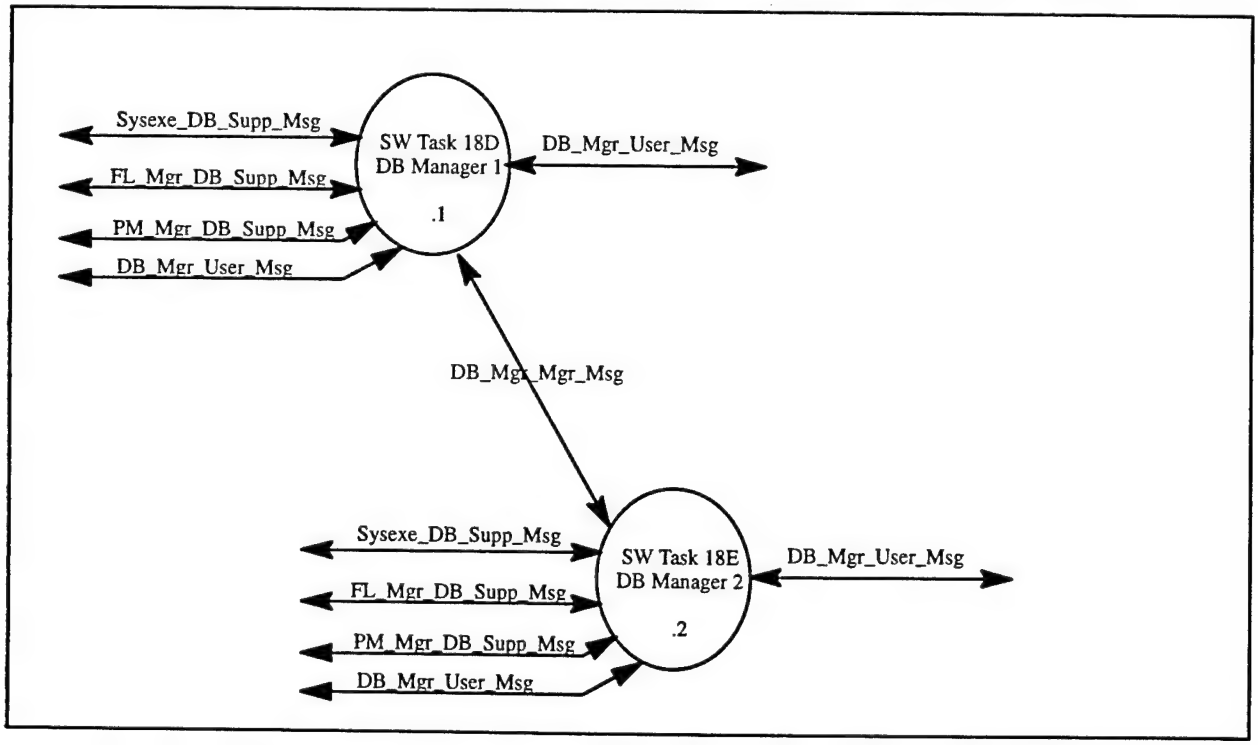



**Diagram 3: Data Base Manager Is Implemented As Two Identical Nodes.**

To provide an example of a DS technique, an algorithm for finding replicated nodes in a graph was developed. This is called the Replication Detection Algorithm (RDA). RDA responds differently according to the given design objective. Suppose the SE has a software design of a passive sonar system [HoKH]. As shown in Diagrams 2 and 3, the software task "Data Base Manager," can be decomposed into two identical software tasks, "SW Task 18D DB Manager 1" and "SW Task 18E

DB Manager 2". If the objective is "to maximize performance," DSA would respond to this particular design by listing the replicated software tasks and giving a recommendation as follows:

```
If these "Tasks" are allocated to the same "Processor," check if
the "Processor" is not over utilized. If so, consider allocating
additional "Processors" to maximize performance.
```

The SE may have a different objective such as "to increase dependability." Then DSA (via RDA) would generate the following message:

```
Consider more "Task" replication to increase reliability. It is
recommended to allocate these "Tasks" to different "Processors"
to improve fault tolerance.
```

## 4. Conclusion

A prototype version of DSA runs under Sun OS 4.1.3. It was implemented using TAE Plus and GNU g++. A sample system design of a passive sonar system [HoKH] was used in testing the DSA prototype version.

DSA will be extended by adding new DS techniques and new functionality over the next year. We have examined adding a new design structuring technique such as ADARTS. Some new DSA functionality such as adopting "user–defined" SDFs will be developed. The subsystems mentioned in the first section will progressively be implemented.

## References


[CCCC94] Computer Command and Control Company, "Design Structuring Assistant: User's Guide (Version 1.1), Contract N60921–93–C–0199, February 7, 1994.

[KiLo] Kim, J. and E. Lock, "Design Structuring for System Engineering", in Proceedings of CSESAW '93, Washington, DC, July 20–22, 1993.

[HoKH] Hoang, N, N. Karangelen, and S. Howell, Mission Critical System Development: Design Views and Their Integration, Version 2.0, Interim Report, Naval Surface Warfare Center, Silver Spring.

[NgHo] Nguyen, C. and S. Howell, System Design Factors: the Essential Ingredients of the System Design, Version 4.0, NSWCDD/TR–92/268, Naval Surface Warfare Center, Silver Spring, MD, March 18, 1994.

# An Approach to Formalize Cooperative Intelligent Information Systems*

Faouzi Boufarès[1], Naoufel Kraïem[2], Faïez Gargouri[3]

1. Laboratoire Informatique Paris Nord, Université de Paris XIII, Av. J.B. Clément, 93430 Villetaneuse, France.Laboratoire
2. MASI/CRI, Université de Paris I - 17 rue Tolbiac, 75013 Paris - France - Phone :(33).1.44.24.93.65
- Fax: (33).1.45.86.76.66 - naoufel@masi.ibp.fr,
3. Laboratoire LIASI IUT de Paris, 143 Ave de Versailles 75016 Paris, France.

**Abstract**

In this paper, we present a multiformalism approach combining knowledge structure and intelligent cooperation, to formalize cooperative information systems. The underlying architecture encourages formalism and tool interoperability. The basic building block is the knowledge base society, specified by the OOPM. Cooperation is managed by an intelligent control, by means of meta knowledge and specific problem solving strategies. Communication relies on service delimitation from internal structure. Dynamics is expressed with production rules in a constraint specification language. A translation method derives colored Petri nets describing system's behavior.

* This work is supported by the Esprit II Project N° 3511 Busniness Class.

**Keywords:** Cooperative information systems, distributed artificial intelligence, reuse, OOPM, decomposition problem solving, knowledge partitioning, validation, colored Petri nets, interoperability, CASE.

## 1. Introduction

Complex systems involve heterogeneous data, shared or exchanged between several modules while achieving a task. So data processing needs to be under control to manage multiple events activating modules' cooperation. At higher level, knowledge must have enough power to deal with processes inherent complexity: non-determinism, parallelism, back-tracking, hierarchical activity structure, and external human intervention. Distributed artificial intelligence techniques [1], [2], [3] offer adequate management means to avoid being overwhelmed by that complexity. For large information systems, features like cooperation, interoperability, distribution, data integration and problem solving are fundamental. The major criticism directed at most actual requirement specification approaches is their poor handling of capturing and modeling knowledge dealing with all those features [4]. These shortcomings can be attributed partly to the inherent nature of the process of capturing, modeling and verifying concepts of the modeled domain and partly to inadequate formalisms used to represent these concepts. Requirement specification implies two basic activities: modeling and analysis. Modeling refers to the mapping of real world phenomena onto basic concepts of a requirement specification language. Analysis refers to techniques used to facilitate the communication between requirement engineers and end-users making use of the requirement specification as the basis of that communication. For future CASE environments supporting the development of complex information systems, the major criteria to meet is to combine the rigor of formal theory, the practicality of existing development methods and the performance analysis of modeling tools in a comprehensive methodology supporting the effective modeling and analysis of that complexity. This need is inherent to the three key-words characterizing future environments: extendibility, integration and broad scope [5], [6]. Extendibility should support the investigation of new process models and the evaluation of novel tools. Integration should enhance extended functionality use and tool cooperation. Moreover, environments should support a wide variety of development activities, and encourage specifications reuse. To support previous requirements, we suggest a multiformalism approach to:

- include different kinds of tools and objects,
- facilitate addition, modification and replacement of components,
- facilitate the cooperation between selected tools and objects.
- support existing models: OMT [7], OOD [8], HOOD [9], ...

The main characteristics of our approach are: modularity, refinement and encapsulation. The modularity is fundamental to describe and manipulate information as individual blocks. The Refinement is a mean to detail abstractions and exhibit pertinent information to be analyzed, whether encapsulation

prevents interference with blocks sharing the same relationship [10].
This paper presents a multiformalism approach to modelize Information Systems (IS) combining knowledge structure with intelligent cooperation. We borrow concepts from object orientation, Colored Petri nets, and distributed artificial intelligent, which make similarities with [11].
However many architectural differences distinguish our suggestion from the previous reference:
At conceptual level, the formalism is object centered using Obejct-Oriented Pivot Model (OOPM) [12]. The basic building block of specification is OBJECT_PIVOT. Dynamic features are expressed with production rules within a constraint specification language.
At logical level, Petri nets are derived from the specified OBJECT_PIVOTs by translations and enrichments [13].
Intelligent cooperation between OBJECT_PIVOTs relies on a clear delimitation of services from communications on one side, and from internal structure on the other side. In particular, the environment is considered as an external OBJECT_PIVOT requiring services from the system (user's request) or offering services to the system (analysis tools, object management, prototyping and simulation).
At the highest level of abstraction, an OBJECT_PIVOT represents a system transition (i.e. activity). Petri net analysis tools and simulators are used to validate specifications against client's requirements and it express the behavior of the objects. Cooperation control to satisfy a given goal is based on two-steps: (i) determine potential OBJECT_PIVOTs societies covering the goal, and (ii) select the ideal subset that will effectively satisfy the goal.

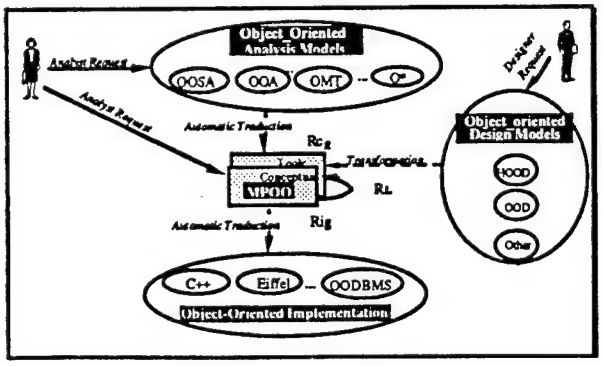



Figure 1. Architecture of Development Environment

The paper is organized as follows: section 2 describes suggested system architecture, based on knowledge partitioning and decomposition problem solving formalized by the OOPM. A short presentation is given in section 3.
The cooperation description is described in section 4. The meta model hierarchy is presented in section 5. Several implementation issues conclude the document.

## 2. An Intelligent Cooperative Information Systems

### 2.1 Motivations

Complex systems cover heterogeneous domains which knowledge and associated representations are processed by complex activities. To handle evolution of the environment, an open framework is crucial: it allows to add new concepts (attributes, methods,...), methods and tools without destroying the general structure. Moreover, it makes module reuse possible along the same resolution process, and supports cooperation of heterogeneous modules [14].
Object-oriented paradigms offer a fundamental modeling framework for systems that support the interaction of loosely coupled, distributed and concurrent executing sub-systems [15]. But validation of obtained systems requires computational models more formal than dataflow diagrams and communicating processes, and specific tools to support both specification and analysis. Ideally, combining object paradigms and formal models for concurrency (in particular Petri nets) is promising, as it encourages user friendliness and reusability of designs with correctness, robustness and efficiency evaluation. However, such multiparadigm approach remains excessively complex [16].
We combine objects and Petri nets through a generic conceptual model called OOPM (§ 2.3), to specify heterogeneous societies(§ 2.2). It is the basic building block of system specification in this paper. It emphasizes the fundamental principles of software engineering: abstraction, modularity, traceability and hierarchy. Its originality is the concurrency.
The main benefit of our approach is operations traceability and weak coupling between software components. These two points constitute a major answer to the software crisis [17].

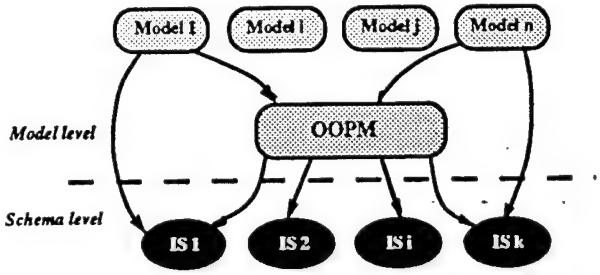



Figure 2: The abstraction levels

## 2.2 Knowledge Architecture

An IS is described as follow : <{I}, KB> where I is an OBJECT_PIVOT specification (§ 3) and KB is Knowledge Base. An IS society is a set of IS describing a same domain.

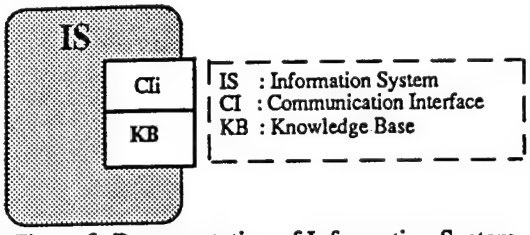



Figure 3: Representation of Information System

A knowledge Base KB is specialized in solving a specific problem. It consists of local production rules (Ri) and a local fact base (Fi).

A society SKB is a set of KBs dexcribing the same application domain. The union of KBi is a partition of the global knowledge base.

An SKB may be organized in two ways:

- its elements may share tasks according to the task-sharing protocol. In this case, a specific module, called manager, allocates a task to each module of the society which is then called worker. It manages activation and termination of operations, according to the well-known client/server model, with the manager as client and the worker as server.
- modules (KBi) may know each other and exchange information according to the result-sharing protocol.

Communication may be horizontal (between KBi into SKB), or vertical (between a KB or an SKB and the control module: module controling the communication protocol).. Communication is based on messages, whose sending and reception is handled by module interfaces MIi.

A SKB communicates with the control module in two cases: to provide execution results, which are put in the shared knowledge space (figure 4), or to request a service from the control, for instance, when it fails to achieve a goal. In such situation, the control has to delegate the considered goal to another society. Communication is supported by the concept of event. Events are used to transcribe pertinent situations (i.e. states) and transitions (state changes). For instance, putting a new result in the shared knowledge space is signaled to the control by an event that formalizes the state change of that space.

The architecture model (figure 4) describes an open information system mainly in terms of the services it offers and those it requires from the environment. It specifies also relations between services according to software engineering standard models [18]. At the highest level of abstraction (i.e. root), offered services formalize system functionality through, for which a user addresses requests. Upon receipt of a request, the system activates adequate processing, based on domain knowledge, noted meta facts, and the reasoning on that domains, expressed by means of meta rules. This information is distributed over cooperative knowledge societies.

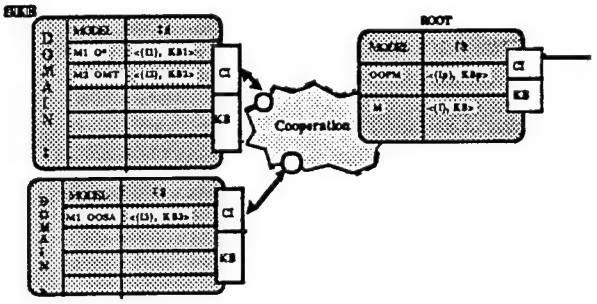



Figure 4. Modeling of Distributed Cooperative Information Systems

The underlying model supports hierarchical descriptions by stepwise service specialization (§ 4). It comes down to reduce service's complexity by defining sub services handling more manageable situations, with major criteria to reduce inferences on a restricted knowledge base defined by its role in the application.

Following sections detail the OOPM. Section 4 details the use of this multiformlism to design of intelligent cooperative knowledge societies.

# 3 The OOPM

The OOPM is sufficiently general to integrate object paradigms and sufficiently formal to allow validation of concurrent behaviors. The later are derived by stepwise refinements of high level models. Formal definition can be found in [19]. The main component of OOPM is OBJECT_PIVOT.

## 3.1 OBJECT_PIVOT Structure

A OBJECT_PIVOT is an abstraction of a real-world entity, which is essentially described by two components: a list of attributes and a list of services.

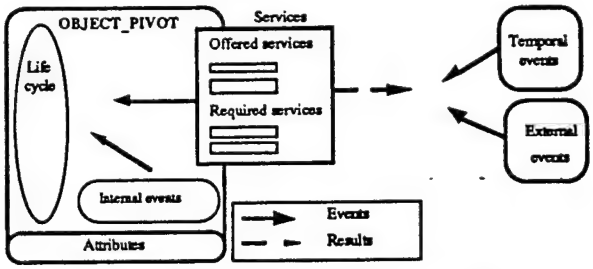



Figure 5. OBJECT_PIVOT Structure

An attribute is an abstraction of a single characteristic possessed by an entity. An attribute may be simple (name string, number: integer) or complex when it

represents a reference to another OBJECT_PIVOT identity. Broadly speaking, attributes are local facts manipulated by the entity. A service is provided by an OBJECT_PIVOT to its environment. Services are divided into provided and required (figure 5). A detailed study of system components responsibilities allows to define the provided and required services.

Services definitions imply several communication features (see figure 5) within a system. These features are commonly known as protocols, and help to structure information flows causally related over time.. Each OBJECT_PIVOT has an identity by mean of which it can be distinguished from other OBJECT_PIVOTs. An OBJECT_PIVOT may be considered at two levels: the type and the instance. The type encapsulates all properties definitions, i.e. attributes, services, relationships and events.
At the instance level, an OBJECT_PIVOT has a state which can change according to some events occurrences. The set of all OBJECT_PIVOT states defines its life cycle. Therefore, a life cycle is related to a type, and each instance has a particular state in that life cycle.
Besides the reference to a type, the formalism allows for sub typing. Sub typing describes a specialization of an OBJECT_PIVOT. Moreover, a type may have several subtypes. This property allows to handle derivation, from a given formalism, of multiple variants sharing type properties, each of which is associated its own properties and proof techniques [20]. The section 5 illustrates the use of the mechanism.
We have retained the *part of* relationship to describe complex domain OBJECT_PIVOTs. This concept allows to express structural links among an OBJECT_PIVOT society whose behavior is strongly coupled. For instance, the management department of a bank is a sub-society of a bank, which has its own services and attributes. However, its structure is a part of the bank structure, and its behavior is strongly coupled to the life of the associated bank. Specifying that subsystem as a standalone OBJECT_PIVOT is not judicious, as the amount of information flows between that OBJECT_PIVOT and the remaining parts of the world is negligible. It is more natural to consider it as a part of the bank.

Services may be accessed by OBJECT_PIVOTs with which the considered unit is involved in a relationship, it may also be accessed by any other OBJECT_PIVOT without underlying structural relationship. For instance, services of an OBJECT_PIVOT Clock may be invoked freely from structural relationships. Such a unit type is close to the standalone object of the TOAD methodology [21].

In the following, we shall briefly demonstrate how an integrate formalization can be organized that enables pivot model. An OBJECT_PIVOT specification I is a tuple $<\Sigma, \Delta>$, where $\Sigma$ is a resource schema formalizing the structural aspects of I and $\Delta$ is diagram formalizing its dynamic aspects (§ 3.2).
An resource schema $\Sigma$ is a tuple <AT, RS, InhS>, where
• AT is the attribute structure. It is the set of all attribute names.
• RS is the relationship structure. It is the set of association and aggregation relationship..
• InhS defines the inheritance structure.
The structural schema $\Sigma$ formalizes the data perspective aspects. It forms the basis for defining an object universe $S_\Sigma$, i.e. a valid population of instances.

## 3.2. OBJECT_PIVOT Behavior

System's behavior is specified in terms of events occurrences and subsequent service activation. A service execution implies elementary or complex transformations applied on some elements of the attribute component. Transformations imply eventual message exchanging to notify a treatment, or an acknowledgment, or a processing result.

### Event

An event occurs when a noteworthy change happens in an OBJECT_PIVOT. The processing to perform this change is embedded in the OBJECT_PIVOT, together with its pre and post conditions. Three kinds of events are distinguished:
- external events stimulated by the environment,
- internal events usually due to an object change,
- temporal events which may occur at certain dates in the system.

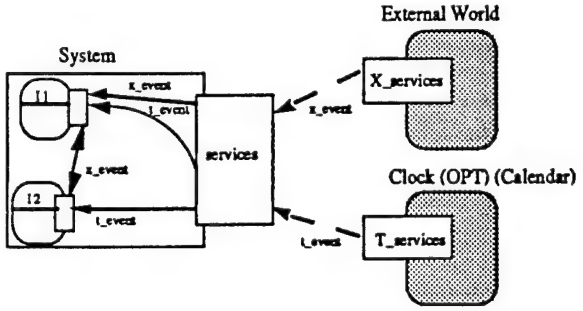



Figure 6. External and Temporal Events

External events may be triggered by society's neighbors or more globally, from the system environment. Temporal events are triggered according to specific schedules specified in a dedicated Temporal OBJECT_PIVOT (OPT).

Action

An action abstracts a set of treatments upon attributes. It may be activated by the occurrence of an event, or by the explicit invocation of a service. An event may trigger one or several actions. A service execution corresponds to one or several actions.

**Message and Service Invocation**

A message formalizes OBJECT_PIVOTs interactions. Interactions correspond to requests for a certain service and associated results, including acknowledgment of receipt and request rejection. An acknowledgment request ensures the effective computation of the required service, while a rejection implies the determination of another instance of OBJECT_PIVOT able to satisfy the request. Information flows are carried within messages. Messages can be grouped under types, to distinguish the different kinds of exchanged information. Each type is characterized by the types of its slots. A slot may contain the sender's identifier, the service name, the list of corresponding inputs and outputs.

The concepts presented above do not give much information about the whole system if considered independently in the specification. Thus, we retained to link them to two formal models: the structural and the behavioral model.

The behavioral model (net schema) depicts services, attributes and their domains, and triggering links between events and services. The services model depicts communications between OBJECT_PIVOTs. Any sending or receipt of a message triggers an event in the system and may change an OBJECT_PIVOT state. The services model is built applying four steps, for each OBJECT_PIVOT:

- identify the set of possible states,
- identify actions which modify a state,
- identify internal, external and temporal events,
- establish triggering links between events and actions.

This process is not always carried out sequentially. Several heuristics can be applied, depending of the amount of available information on the domain:

- the specifier may determine actions first, and then define the pertinent states of the system as those affected by action executions. The model may be enriched by refining existing states or adding new ones after a check of completeness. The approach is called action-driven.
- another approach consists of describing the main states of the system and associated event types (i.e. list of possible events occurring at those states). Actions are then determined as executing state changes.

In practice, complex applications use hybrid approaches [22]. This is often due to the lack of information (incomplete requirements), i.e. to the difficulty to express clearly events and states in some components, and to predict

The net schema $\Delta$ represents the dynamics of the system in the form of resource network N consisting of business actions that manipulate objects in $S_\Sigma$.

Let $\Delta$ be a net schema for a resource schema $\Sigma$ with an associated resource universe $S_\Sigma$.:

$$\Delta = <P, T, M_0, C>$$

- P is the set of places;
- T is the set of transitions. Each transition in this set represents an action;
- $M_0 \in S_\Sigma$ is the initial marking for the net;
- C is a set of of colors : $C = \{C_1, C_2, ..., C_n\}$. Where a color is defined as follows: $C_k = <c_{k1}, c_{k2}, ..., c_{kn}>$.

## 3.3 OBJECT_PIVOT Validation

At a given level of abstraction, the validation and verification of an OBJECT_PIVOTs concerns:

- the static information (data types) on which completeness and coherency checks are achieved as for abstract data types. In particular, it is checked whether operations defined on data do not violate their integrity constraints. We do not develop such verification since they are classical in literature.
- behavioral information handled by associated nets. Qualitative as well as quantitative results are provided on nets [23]. For instance, it can be found that a certain structure of life cycle leads to deadlock, or that a given message can never reach its destination, without having to simulate the net. Furthermore, it can be calculated the mean number of messages (tokens in nets) hold by an OBJECT_PIVOT, or waiting for processing in a given queue (place of the net). Moreover, timed and stochastic nets are also analyzed which provides an interesting evaluation of components performances (e.g. mean time of message processing, or of service answering, etc.).

Globally, the modeling process emphasizes following properties:

- coverage of target system *activities*: through OBJECT_PIVOTs properties, all system's activities should be covered, even through abstract actions that will be refined.
- behavioral coherency: for cooperative OBJECT_PIVOTs, the set of associated events and messages should be coherent with respect to their life cycles.
- activities independence: units should cover independent activities of the target system, eventually synchronized with others on shared resources or shared events.

- preservation of dynamic transitions: for a complex service that triggers further actions or services on several components, the atomicity of the whole transition should be guaranteed.

## 4. On the Design of Intelligent Cooperative Societies

### 4.1 Knowledge Partitioning Criteria

The underlying system architecture (figure 4) is based on the modularization of information representation and analysis of knowledge base societies, by means of the OBJECT_PIVOT formalism. The major advantage is the possibility to specialize information into autonomous modules, to reduce the complexity and inherent time responses in complex problem solving situations.
These advantages are visible both at the design and the implementation levels:
Modular design is based on the concept of *knowledge packets*, and the associated control and communication mechanisms, necessary to the cooperation in the system. Knowledge specialization delimits the activation part of the services at their interface according to the object paradigm [8]. This facilitates packets' validation and reuse, and allow to define the cooperation protocol between these knowledge pakets independently from the formalisms used for they implementation.
The complexity of system is reduced when decomposing it into more manageable tasks [24]. The control mechanism allows then to detect as early as possible, the part of the system responsible for a sub-problem, i.e. *representation* and *validation* of a certain view- and focus only on this part, called knowledge packet.
The cooperation protocol synthesizes the results of sub-problem resolutions, and presents them to the end-user interface, relying on the degree of interoperability allowed by the system.
The concept of selection of the adequate formalism dealing with a given sub-system is ensured by the generalization of an artificial intelligence technique, the *high level pattern matching* [25], where two levels are distinguished: the classical pattern matching dealing with simple production rules, and a meta pattern matching whose granularity is not any more the rule, but a packet of rules. This distinction allows to reduce the complexity of a problem solving, and specifically, the impact of inferences on a restricted knowledge base i.e. the amount of information necessary to handle a given sub-problem.
From a validation point of view, object-orientation has shown to be very promising. However, the possibility of validating autonomous components either by state machines or communicating processes has also shown the limits of selected formalism, in the inherent concurrency of modeled entities. Therefore, concurrent formalisms were needed [16]. This motivated the integration of colored Petri nets in the OOPM.

### 4.2 Description of the Knowledge Schema

The approach consists then of structuring an information system as a society of knowledge granules, encapsulated in OBJECT_PIVOTs, characterized by hierarchical interdependencies, and processed by specific rules. More details are given in [30].
OBJECT_PIVOT properties are mainly described by static properties, (facts and/or meta facts), and a behavior expressed dynamically with production rules. Production rules are modeled and validated with colored Petri nets. Facts and meta facts are used by the inference engine, and processed by the production rule language.
The Knowledge structure is defined according to a typology which induces the role of the different modules in a problem resolution process. However, a challenger solution necessitates more than efficient control strategy. It requires the selection of high performance representations of the problem, and efficient paths and conditions.
The difficulty in using declarative knowledge is proportional to their declarative aspects: it is easy to saturate a set of rules to achieve a goal, but it is difficult to predict the amount of attempts the system must do before reaching it. Minimizing this amounts is ensured by defining orders on rules, i.e. define a meta control mechanism on rules.

## 5 The Meta Model Hierarchy

The main of this section is to define a correspondence between the pivot model and other models. The pivot meta model describe the set of meta concepts of the OOCPM.
The Meta Model Hierarchy, being proposed in [Kraïem94], is a framework relating modeling techniques on basis on their meta models. Due to the evergrowing amount of coexisting nowadays, it is unrealistic to construct a complete Meta Model Hierarchy.
In the Meta Model Hierarchy a meta model is composed of a concept structure and a set of constraints restricting the possible instanciations of the meta model. The concept structure consists a set of concepts, and relationships between these concepts.
The intention of this section is to illustrate the practical use the Meta Model Hierarchy. The techniques Entity Modelling (ERM), NIAM and OMT will be positioned in the Meta Model Hierarchy following the top-down approach [Oei92]. Each node in the hierarchy correspond to a meta model plus the set of meta models

which can derived from it. Meta models on one path in the hierarchy are upward compatible.

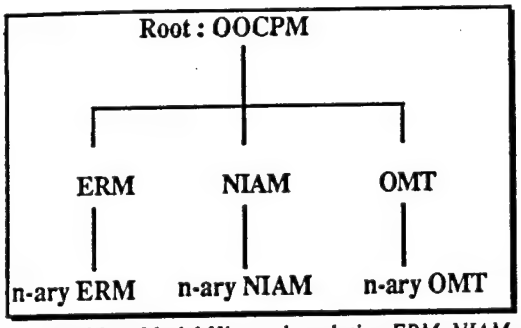



*Fig. 7: The Meta Model Hierarchy relating ERM, NIAM and OMT*

To construct a part of the Meta Model Hierarchy which relates the meta models of the techniques to be considered, a particular meta model is chosen to become the root of this part of the hierarchy. The choice for the root has to be a meta model from which it is possible to end up in all the target meta models by an appropriate set of partitionings, restrictions and/or degenerations [Oei92]. Note that the root of the Meta Model Hierarchy has to be a meta model which has the same or more expressive power than all the other meta models of the hierarchy.

In this example, the root that is chosen is a simplified OOCPM which describe object, message, operation and relationships. This root suffices to relate the (simplified) meta models of the techniques which will be taken into consideration in this example.

In order to relate the meta models of NIAM, ERM, and OMT in the Meta Model Hierarchy the following procedure is chosen. First of all, the meta model of NIAM is related to the root meta model by a sequence of partitionings, restrictions, and degeneration. In this process some intermediate meta models are constructed. After the construction of this path, the meta model of ERM is connected by applying Meta Model Hierarchy operators on one of these intermediate meta models on this path. Finally, the meta model of Petri-Nets is incorporated in the Meta Model Hierarchy by a set of appropriate Meta Model Hierarchy operations.

## 6. Modular Implementation Issues

We discuss in this part the impact of the suggested methodology on the design and implementation of intelligent cooperative information systems.

At the design phase, several features are attractive:

- The design of knowledge modules specialized in the resolution of specific problems and their incremental integration in the knowledge base.
- concurrent activation of modules to deal with specific tasks, or sequences of activation for complex tasks. The advantage is to be able to reason at the adequate abstraction level which is the task level, to effectuate coherently complex tasks.
- Module reuse along the same resolution process is possible, by means of a systematic representation transformation. The advantage is the inheritance of properties of a given model from another model, and the use of tools defined on other representations (other than current representation).
- Knowledge modules cooperation independently from their application domains.

At the implementation level, the approach encourages:

- Reduction of problem complexity, by decomposition and invocation of the society of modules specialized in solving each sub-problem. It reduces considerably time responses, provided that the resolution process is guided by goals.
- For each problem solving process, reduction of the impact of inferences on a restricted knowledge base necessary to the current problem. This reduces, besides time responses, memory allocated to knowledge at a given moment.
- The possibility to focus on a goal, by a backward chaining on necessary sub-goals, which avoid inferring on other goals deductible when using a forward chaining.

The multiformalism approach constitutes an attractive step towards physical distribution of IS knowledge. The structure of OBJECT_PIVOT induces several guidelines on the architectural features necessary to support of distribution and the cooperation of the specified systems.

## 7. Conclusion

In this paper, we presented a specification methodology combining object orientation, distributed artificial intelligence and concurrent paradigms, to formalize cooperative information systems.

The approach takes inspiration from recent integration streams animated by requirements for future development environments [28]. It emphasizes user friendliness and reusability of designs with the correctness, robustness and efficiency evaluation. The involvement of large agreat amount of knowledge, shared or exchanged between several components, while achieving a task needs efficient control, to manage multiple events activating modules cooperation. At the higher abstraction level, knowledge must have power enough to deal with processes inherent complexity: non-determinism, parallelism, back-tracking, hierarchical activity structure, and external human intervention. To avoid being overwhelmed by this complexity, the approach suggests a meta control strategy based on a hierarchical representation of knowledge, the definition of services

processing representations, and the definition of meta knowledge necessary to plan component activation -in particular, verify whether the specification can be executed or not in a given context- and control their cooperation on complex tasks. At component level, the suggested formalism ensures analysis and validation of behaviour by translation of dynamic concepts into colored Petri nets. Furthermore, specification translation could be envisaged into modular colored nets, to preserve the modularity provided by high level specifications [29]. We are currently working on the theoretical features of adequate modular nets, focusing on their compositionality and reduction features.

## 7. Bibliography


[1] C. Hewitt, "viewing control structures as patterns of passing messages", Artificial Intelligence, vol. 8, pp 323-364, 1977.

[2] E.H. Durfee, V.R. Lesser, D.D. Corkill, Cooperative Distributed Probem Solving, in A. Barr, P.R. Cohen and E.A Feigenbaum, editors, The Handbook of Artificial Intelligence Vol. 4, Addison Wesley 1989, pp. 83-147.

[3] R. G. Smith, "A Framework for problem solving in a distibuted processing environment", Dep. Comput. Sci. , Stanford Univ., Stanford, CA, Stan-CS-78-700 (HPP-78-28) Dec.1978.

[4] J.R. Abrial, On Constructing Large Software, in Proc. of the 12th World Computer Congress IFIP'92 "From research to practice", Volume 1, pp 103- 112, (North Holland), Madrid, Spain, September 1992.

[5] J. Lonchamp, C.Godart, J.C. Derniame, Les environnements intégrés de Production de Logiciel, in Technique et Science Informatiques, Vol 11- n°1, pp 31-95, Hermes 1992.

[6] J.C. Wieleden, A.L. Wolf, W.R. Rosenblatt, P.L.Tarr, Specification Level Interoperability, in CACM, May 1991, Vol 34, N° 5.

[7] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modeling and Design", Prentice Hall, 1991

[8] G. Booch, Object-Oriented Development with application, The Benjamin Cummings Series in Ada and Software Engineering, (1991).

[9] M. Heitz, "HOOD, une méthode de conception hiérarchisée orientée objet pour le développement des gros logiciels techniques et temps réel", Déc. 1989, Journées Ada-France.

[10] D. Parnas, On the criteria to be used in decomposing systems into modules, commun. ACM, 15, 12 (1972), 1035-1058.

[11] C. Hoffman, B. Krämer, B. Dinler, Multiparadigm Description of System Development Processes, in LNCS 635, Process Technology, pp.123-137, J.C Derniame Ed., 1992.

[12] N. Kraïem, F. Gargouri, F. Boufares, "From Object-Oriented Design Towards Object-Oriented Programming", in the Proceedings of the Fifth Conference on Advanced Information Systems Engineering (CAiSE'93), EEC, Paris, Juin 1993.

[13] N. Kraïem, F. Boufares, "A Generic Object Oriented Conceptual Pivot Model", in the Proceedings of the 1993 Complex Systems Engineering Synthesis and Assessment Technology Workshop (CSESAW'93), Washington, USA, Juillet 1993.

[14] H. Mili, "Intelligent Component Retrieval for Software Reuse", in Proc. of MCSEAI'94, April 1994, Maroc.

[15] B. Hendersen-Sellers, J.M. Edwards, Object-Oriented Systems Life Cycle, CACM, Vol.33 N°.9 (1990).

[16] P.Wegner, Concepts and Paradigms of Object-Oriented Programming, Expansion of Oct 4 OOPSLA-89 Keynote Talk, S. G. P. L. Publication, ACM Press.

[17] B. Meyer, Object-Oriented Software Construction, Prentice Hall, Hemel Hemstead, 1988.

[18] A. Earl, A Reference Model for Computer Assited Software Engineering Environment Frameworeks, Hewlett Packard, Technical Memo, V. 3.0 ECMA/TC33/TGRM/90/011, May 90.

[19] N.Kraïem, J. Brunet " Mapping of Conceptual Specifications into Object-oriented Programs", SEKE'92, Proceedings of the Fourth International Conference on Software Engineering and Knowledge Engineering, IEEE, Capri, Juin 1992.

[20] H. Bachatène, Modeling_Units: A High Level Formalism To Specify Complex Distributed Systems, Proc. of 7th ISCIS International Symposium on Infomation Systems, Antalya, Turkey, Novemeber 1992.

[21] Chean-Wee Chee and al, Toad : Towards an Object-Oriented Analysis and Design Methodology : Experiences & Preliminary Observations", in Proc. of TOOLS 90, (1990).

[22] J. Brunet, Modelling the World with Semantic Objetcs, Proc. of the Working Conference on The Object Oriented Approach in Information Systems, Quebec (1991).

[23] High-level Petri Nets, Theory and Application, K. Jensen, G. Rosenberg Eds.April (1991).

[24] R. Davis, R.G. Smith, Negotiation as a Metaphor for Distributed Problem Solving, in Artificial Intellingence20, pp 63-109, North Holland, (1983).

[25] A. El Fallah Seghrouchni, A Knowledge base Architecture for Petri Nets Analysis, Ph.D, Blaise Pascal Institute, University Paris VI, Paris, December (1991).

[26] N. Kraïem, "Conception d'un Modèle Pivot Orienté Objet", Ph.D, Blaise Pascal Institute, University Paris VI, Paris, (1994).

[27] P. Estraillier, C. Girault, Applying Petri Net Theory to the Modeling, Analysis and Prototyping of Distributed Systems, in Proc. of the IEEE/SICE Int.Workshop on Emerging Technologies For Factory Automation: State of the Art and Future Directions, Australia, August (1992).

[28] Computer Science and Technology Board, Scaling Up: A Research Agenda for Software Engineering, Communications of the ACM, Volume 33, N° 3 (1990).

[29] I. Bernadinello, F. DeCindio, A Survey of basic net models and modular net, advances in Petri nets, G. Rozenberg, Ed. April (1992).

[30] H. Bachatène, A El Fallah, A Multiformalism Approach To Formalize Intelligent Cooperative Information Systems, ICICIS93, The Netherlands 1993.

# Distributed Synthesis Tools for Mission-Critical Computer Systems

Parameswaran Ramanathan  Ahmad Abualsamid

Department of Electrical and Computer Engineering
University of Wisconsin–Madison
Madison, WI 53706–1691.
parmesh@ece.wisc.edu, (608) 263–0557

At University of Wisconsin–Madison, we have been developing a synthesis system called SHARMA (Synthesis of Heterogeneous Architectures for Real-time Mission-Critical Applications), whose objective is to facilitate design of computer systems for mission-critical applications. The focus of this system is on synthesizing heterogeneous distributed computing systems which meet the constraints of a given mission-critical application.

SHARMA is comprised of an integrated set of tools for carrying out the steps in the synthesis process. An overview of these tools was presented at CSESAW'93 [1]. These tools have been used to identify low-cost architectures for several synthetic applications. Our experience with these synthetic applications indicated that tools have one serious limitation. The number of design alternatives is so large that the tools require an enormous computation time on a single workstation to identify a suitable low-cost architecture. Due to the inherent complexity of the mission-critical applications, this limitation is not unique to SHARMA, but, common to most synthesis tools.

One way to alleviate this limitation is to make use of the computing resources in a network of workstations to speed up the search and evaluation of design alternatives. Since network of workstations have become commonplace in most design environments, this is an attractive solution. Furthermore, the granularity of the parallelism in the synthesis algorithms makes them well suited for distributed execution on a network of workstations. As a result, we have been pursuing this approach for the past one year and the work is still in its initial stages. In this paper, we outline our approach for building a set of tools which can be run in distributed fashion on a network of workstations. We begin with a brief overview of the tools in SHARMA and then present an example to illustrate our approach.

Figure 1 shows the operating environment of our synthesis system. The two inputs to the synthesis system are a design library of components and the mission-critical application. The output includes a heterogeneous architecture containing components from the design library, a mapping of the tasks in the application onto the modules in the architecture, and a schedule for the tasks and messages in the application. The design library contains processors, resources, and interconnects from which the distributed system is to be synthesized.

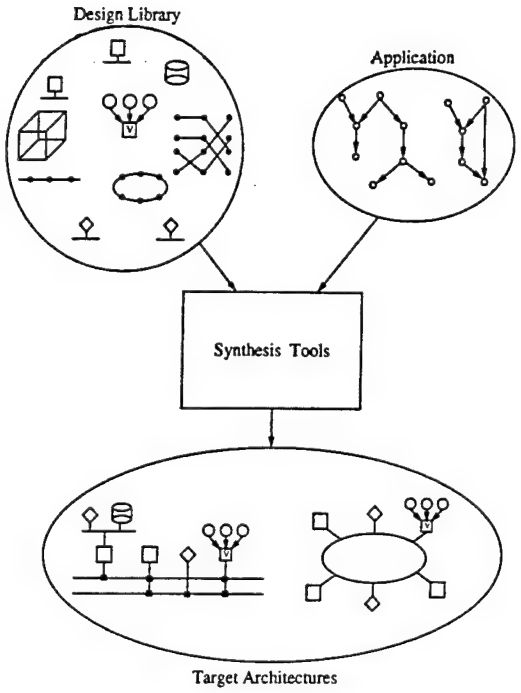



Figure 1: Operating environment of the synthesis system.

Components in the design library are characterized by attributes such as performance and cost. The application is specified as a set of cooperating tasks and the constraints that each task must satisfy, e.g., a task may have deadline constraints, resource requirements, precedence relations, computational needs, etc. The objective is to identify the lowest cost computer system (constructed using the components in the design library) which satisfies all the constraints of the application.

Figure 2 shows the key steps in the synthesis process. Steps 2–4 need to be parallelized to make effective use of the network of workstations. Although we have developed some parallel algorithms for each of these steps, in this paper, we will only focus on our approach for parallelizing Step 4.

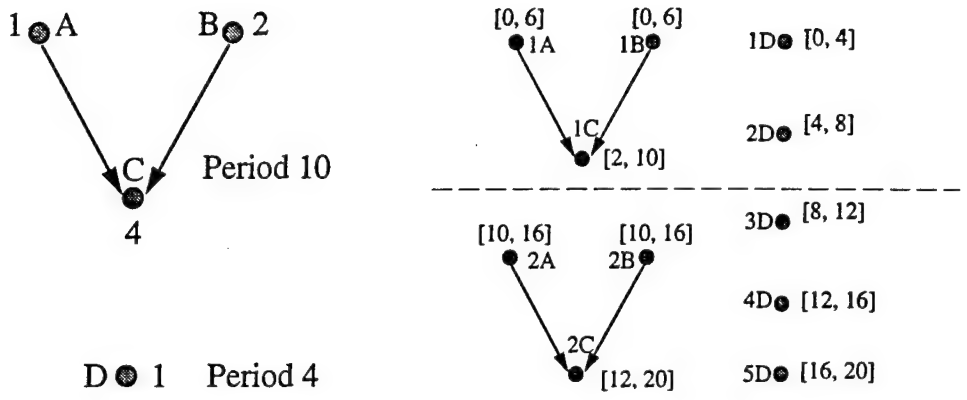
In Step 4, the application tasks are scheduled on a candidate architecture identified in Step 3. Our approach to parallelizing this step involves partitioning the application tasks into smaller subsets and assigning a subset to each workstation. The workstation is then responsible for scheduling the tasks (and the resulting messages) in the subset allocated to it. The main advantage of this approach is that a workstation has to deal with only a smaller number of tasks. However, unless care is taken while partitioning the application tasks, the amount of communication required to coordinate the scheduling process may be quite large. In the rest of this paper, we present a simple example to illustrate the need for a suitable partitioning and outline our approach to identify such a partition.

1. Input the mission-critical application and the design library.
2. Compute a lower bound on the number of processors, resources, and interconnects required to meet the constraints of the application.
3. (a) Choose processors and resources from the design library for inclusion in the architecture.
   (b) Identify a suitable interconnection structure between the selected components.
4. Evaluate the architecture identified in Step 3 by scheduling the application tasks and the resulting messages on the architecture.
5. Terminate if a satisfactory solution is found. Otherwise, go to Step 3 and improve the architecture based on the results from Step 4.

Figure 2: Key steps in the synthesis process.

Consider the simple application shown in Figure 3 (a). The application has two periodic jobs with periods 4 and 10, respectively. The first job is comprised of three non-preemptive tasks (namely A, B, and C) whereas the second periodic job has only non-preemptive task (namely D). The precedence relations between the tasks in the first job are shown as directed arrows. The execution time of the tasks is shown as weights near the corresponding vertices. Figure 3(b) shows the invocations of these two jobs in a superperiod (i.e., least common multiple of the periods). The scheduling heuristic identifies a static schedule for all the tasks in this superperiod and repeats the schedule for the subsequent superperiods. The numbers enclosed in [ ] are the earliest start time and the latest completion time of the corresponding task, ignoring the time required for communication.

Now suppose that this application is to be scheduled using two workstations. Then, we must partition the tasks in Figure 3(b) into two groups and assign them to the two workstations. For example, all tasks above the dotted line can be assigned to one workstation while those below the dotted line can be assigned to a different workstation. If the workstations do not coordinate while scheduling their respective tasks, then two distinct tasks may get scheduled on the same processor at the same time, e.g., the time interval $[8, 9]$ may be part of the scheduled time on the same processor for tasks 1C and 3D because they are being handled by two different workstations. This is clearly an invalid schedule for the application. Our approach for preventing such invalid schedules is to partition the task set such that latest start time of all tasks in a partition are less than the earliest start




(a) Periodic jobs

(b) Invocations in a superperiod

Figure 3: An example application.

time of all tasks in the other partition[1]. If no such partition exists, then we can impose a release time constraint on some tasks to attain this property, e.g., the partition determined by the dotted line will satisfy this property if we impose a release time of 10 on task 3D. Such restrictions may eliminate some feasible schedules from consideration and thus reduce the chances of finding a feasible solution. However, this is not likely to be a serious problem in large applications. This problem can be further alleviated by partitioning the graph in such a way that as little restriction as possible is placed on the tasks.

The problem of identifying a partition with minimal restriction can be formulated as follows. Let $G$ be a weighted undirected graph in which the vertices correspond to the tasks in the application. There is an edge between vertices $i$ and $j$ iff $[est_i, lct_i] \cap [est_j, lct_j] \neq \emptyset$, where $est_i$ and $lct_i$ (respectively, $est_j$ and $lct_j$) denote the earliest start time and the latest completion of task $i$ (respectively, task $j$). The weight associated with an edge $\{i, j\}$ is $\infty$ if $[est_i, lct_i] \subseteq [est_j, lct_j]$ or vice versa. Otherwise, it is equal to the amount of the intersection between $[est_i, lct_i]$ and $[est_j, lct_j]$. The partitioning problem then is to identify two groups with approximately the same number of vertices such that the sum of the weights of cross-edges is minimum.

Figure 4 shows the weighted undirected graph for the application in Figure 3(b). This graph can be divided into two groups with only one cross-edge as shown by the dotted circles. From this cross-edge one can conclude that by imposing a release time of $8 + 2 = 10$ on task 3D, all tasks in the upper partition will have latest completion times less than the earliest start times of all tasks in the lower partition. With this additional timing constraint, the two workstations can schedule the subset allocated to them without any coordination.

[1] A slightly modified condition is required if there are more than two partitions.

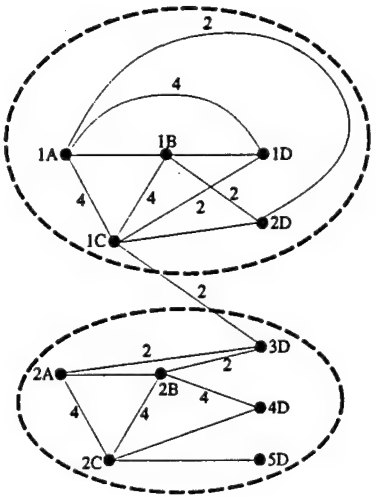



Figure 4: Partitioning graph for the application in Figure 3.

This is because the possible schedule times for these two partitions are disjoint in time.

Several sequential and parallel heuristics have been proposed in literature for solving the above partitioning problem [2,3]. We are currently comparing the performance of these heuristics. Results of these comparisons will be presented at the workshop.

## References

[1] R. Alqadi and P. Ramanathan, "Architectural synthesis of mission-critical computing systems," in *Proceedings Complex Systems Engineering Synthesis and Assessment Technology Workshop*, pp. 185–192. Naval Surface Warfare Center, Silver Spring, Maryland, July 1993.

[2] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *AT&T Bell Labs Technical Journal*, vol. 49, pp. 291–307, February 1970.

[3] J. E. Savage and M. G. Wolka, "Parallelism in graph partitioning," *Journal of parallel and distributed computing*, vol. 13, pp. 257–272, 1991.

# Complex System Optimization

J. Pukite and P.R. Pukite
DAINA
4111 Central Avenue NE, Suite 212
Columbia Heights, MN 55421-2953
(612)781-7600
pukite@daina.com

**Abstract**

The designer of a complex system needs to select the best system configuration that meets the specified design requirements, design constraints, and the established cost goals. This task, usually called system optimization, is rather complex because of the extremely large number of potential design options. Design optimization complexity further increases if component redundancy is required to meet the required system reliability goals. This increase, due to the evolution of new system mechanization and redundancy techniques, requires the designer to consider alternate configurations which any design might take and still meet the prescribed system requirements. The optimization problem complexity will also increase at an exponential rate with the number of subsystems. In these situations, optimization requires a great amount of ingenuity, effort, and experience on the part of the system designer. Because of this complexity, full-scale system design optimization is seldom attempted and the final system is less than optimal.

It is obvious that the practical optimization problem is not easily solvable using manual techniques and that computer aids are needed. Unfortunately, very few practical optimization procedures and programs have been developed, although numerous theoretical investigations concerning the foundations of optimization exist. In this paper we will describe one computer-based optimization program that is easy to use and is applicable to a variety of systems.

## 1.0 An Overview of the Optimization Process

An overview of the role of optimization in a system design process is presented in Figure 1. This figure shows the major top-level tasks involved in optimization as well as the iterative loops. During a design process, these loops will be traversed several times. The primary loop (1) represents the definition of the optimization criteria. The inner loop (2) represents the actual optimization process. During the early design stages, an optimal solution – based on the given constraints – may not be found. In these situations some of these constraints may have to be loosened. This path is represented by the contingency loop (3).

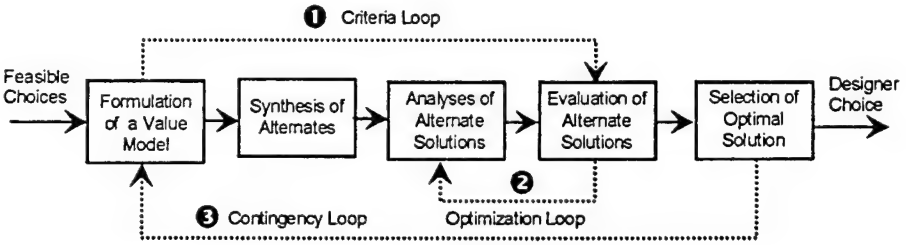



**FIGURE 1. Complex System Design Process**

Based on the specific system design requirements, individual optimization criteria are selected. These will include performance, cost, and probability bases. In the following discussion, we will use these terms in a wider sense. Thus, the general cost basis will not only include dollars, but also physical parameters, such as weight, power, volume, *etc.* Similarly, the probability family of optimization criteria will include the various failure probability constraints.

Redundant system design trade-offs will be conducted at several levels. These trade-offs will involve taking any portion, large or small, of a given system, to evaluate the differences between the originally conceived configuration and any proposed alternative. Since most of the subsystems will be highly interrelated, and the design constraints are normally expressed at the system level, the final trade-offs should always be conducted at the system level.

## 2.0 Trade-off Model

The derivation of a trade-off model arises from the need to compare various alternate mechanizations so that the best mechanization can be selected within the established design constraints. The trade-off model describes the alternate mechanizations in terms of their parameters such as performance, reliability, maintainability, complexity, size, weight, power, and cost. The trade-off model must also identify the importance of each of these parameters in the proposed system. Thus, there is no formula that will solve all trade-off mechanization problems. Each case must select and generate its own criteria.

When conducting a trade-off analysis, the lowest level partitioning usually will be a subsystem. Each of these subsystems may have an arbitrary number of alternate configurations, each with different performance parameters, costs, reliabilities, *etc.* As a first approximation, we assume that the system consists of independent subsystems, as shown in Figure 2. We also note that the number of feasible system configurations can be large, as shown in the example block in this figure.

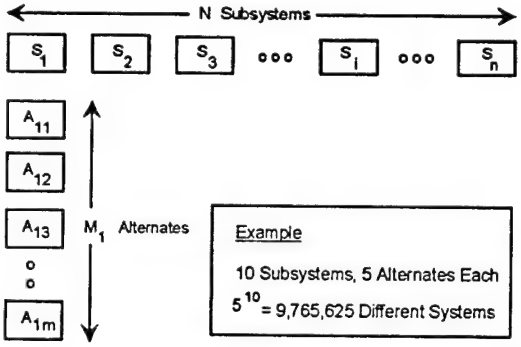



**FIGURE 2. System Optimization Problem**

## 3.0 Optimization

To claim an optimum system infers that we have attained perfection in some way. It also implies that all of the possible solutions have been investigated and that the best solution has been found. Unfortunately, this ideal case cannot be reached in real life. However, we should investigate as many solutions as is reasonable and practical and to select the best from the available choices (although this is not a simple task).

The optimization techniques can be divided into two groups: *discrete* and *continuous*. In engineering problems, the discrete cases are the most important because they are imposed by the discrete nature (structure) of the systems and usually involve several discrete decision variables. The optimization of discrete systems can be reduced to a stage-by-stage decision process where the optimum values are selected at each stage so that the final objective function of the system is maximized within the given constraints.

Two of the key discrete optimization techniques are: *dynamic programming* [Bellman 57, Aris 64] and *discrete optimum principle* [Fan 64]. Although these techniques use different problem formulations, the results obtained normally will be similar.

Dynamic programming is based on the "Principle of Optimality" [Bellman 57, p.83]:

> *"An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."*

A paraphrase of the above is given by Aris [Aris 64, p.27]:

> *"It is really saying that if you don't do the best you can with what you happen to have got, you'll never do the best you might have done with what you should have had."*

At present, dynamic programming represents the most powerful method for solving system configuration optimization problems of the discrete type. Dynamic programming uses a stage-wise approach where the sub-optimum configurations are rejected at each stage to prevent them from entering the next stage in the computation process.

A basic concept in dynamic programming is the determination of the *dominating sequence*. This concept can be best understood by examining Figure 3. This figure represents a two-dimensional plot of a number of alternate system configurations.

The dominated configurations are represented by stars and the dominating sequence by numbers in solid circles. We note that a dominated configuration in this case is one for which there is another competing configuration which has higher reliability and/or lower cost.

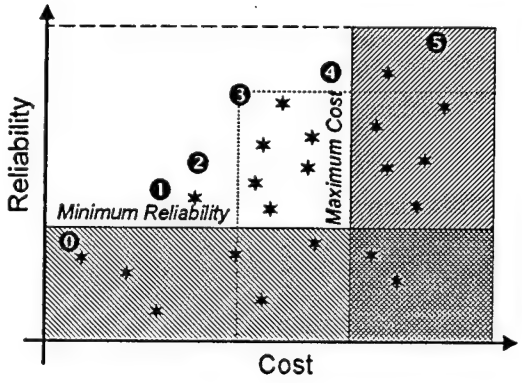



**FIGURE 3. Dominating Sequence**

For example, the dominating configuration (3) dominates all of those configurations that are enclosed by the dotted lines. In this example, the dominating sequence includes configurations 1 through 4. We are rejecting configuration (0) because it does not meet the minimum reliability requirements, and configuration (5) because it exceeds the maximum cost limit. If a configuration cannot be selected from the given feasible choices, then two alternatives should be considered: (a) the system constraints should be reevaluated to determine if the given constraints could be relaxed, and (b) the feasibility of including some alternate subsystems considered that were not evaluated before should be evaluated.

*Auxiliary constraints*. The conventional optimization process considers only two competing design variables at a time. However, it is possible to consider additional variables during the optimization. These auxiliary variables usually represent system imposed constraints which cannot be exceeded.

An example of how an auxiliary variable affects the optimization process is illustrated in Figure 4. In this case, we are extending the previous reliability versus cost optimization problem to include power as an auxiliary variable (with a given maximum power constraint). In this optimization situation, the feasible solution region is bounded by the constraint planes representing reliability, cost and power. Thus, the dominating configurations cover a three-dimensional space.

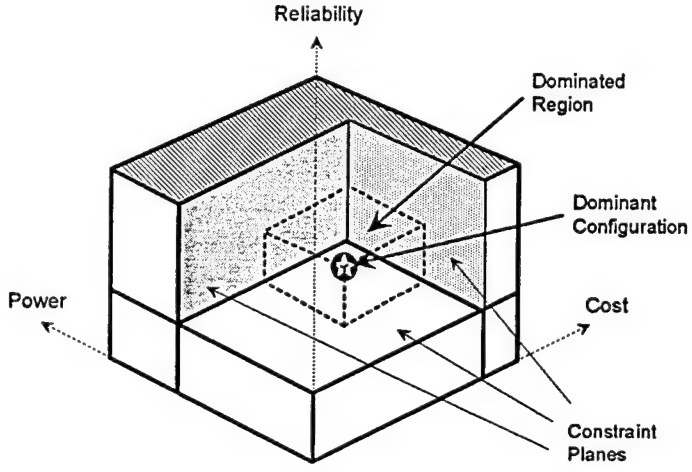



**FIGURE 4. Auxiliary Constraints in Optimization**

*Feasibility of a solution.* Not every optimization problem has a solution. If a solution cannot be found at the system level, then other alternatives must be examined or system constraints relaxed.

## 4.0 Computer-Aided Optimization

Even a simple system design problem may yield many potential system configurations. In our earlier example, shown in Figure 3, 10 subsystems with 5 alternates each, resulted in almost 10 million potential configurations, clearly out of the range of manual evaluation. Similarly, the Voyager spacecraft system, that was designed two decades ago, consisted of 166 different assemblies grouped into 51 functional families. When considering the different redundancy implementations, this resulted in $10^{21}$ potential system configurations and its optimization required computer support. Systems that are designed now, for use in the next generation applications, are even more complex and their optimization requires a major effort from the designer.

In these situations we have to resort to computer-aided optimization programs. One such program is ***CEO*** – *Cost Effectiveness Optimizer*– an experimental program developed by DAINA. ***CEO*** can be used to optimize systems which consist of a number of independent subsystems, with the basic configuration shown in Figure 3. It is a interactive program with a simple interface and it operates under the Microsoft Windows™ environment.

***CEO*** is dialog-oriented and pushbutton-driven. The operational sequence is shown in Figure 5 and involves three different dialog windows.

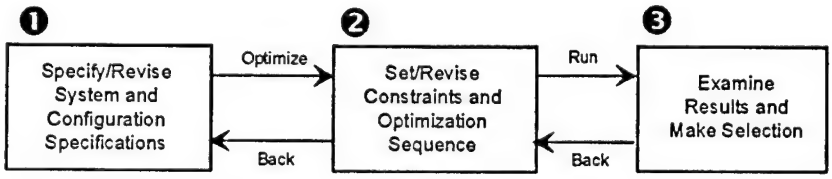



**FIGURE 5. CEO Operational Sequence**

In the *Define Subsystems and Configurations* Window (Figure 6) we define the various subsystems and their configurations. This window is also used to enter the configuration data.

In the example shown, there are four different subsystems and the selected subsystem – `eca (yaw)` – has eight potential candidates (shown in *Configuration Names* list box). The highlighted text represents the current user selection.

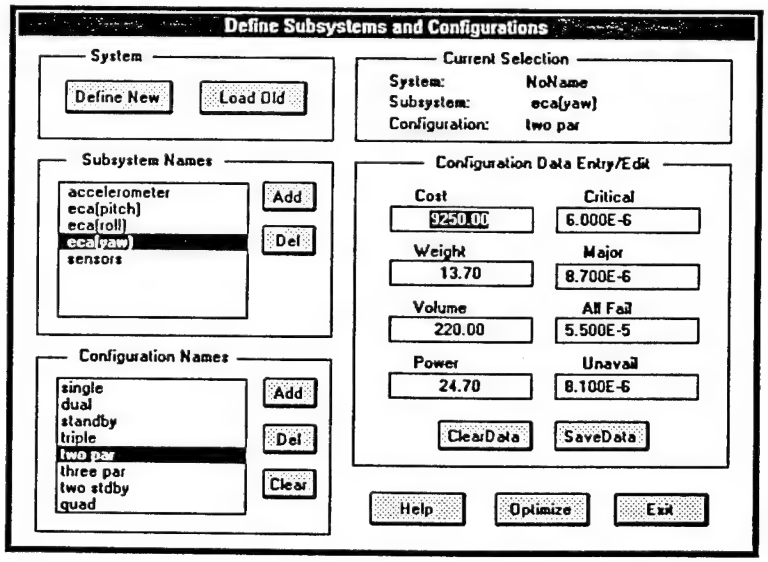



**FIGURE 6. CEO Define Subsystems and Configuration Window**

In the *Set Optimization Sequence and Constraints* Window (Figure 7), we select an optimization sequence using radio buttons and enter system constraints.

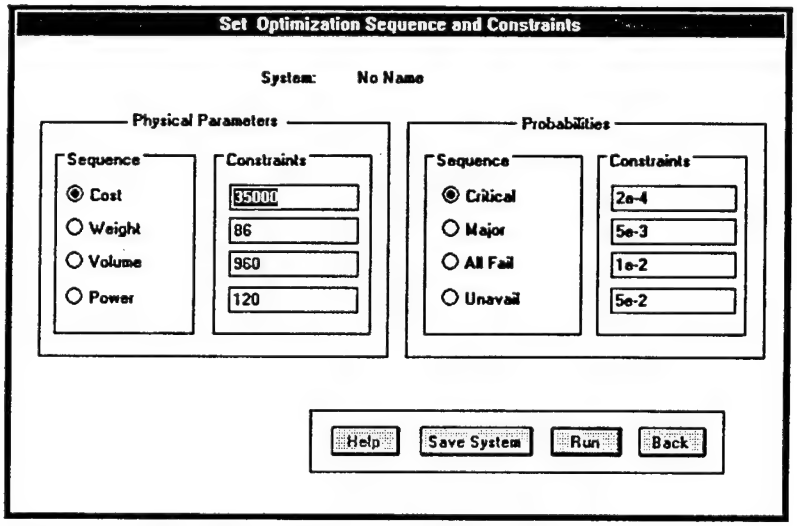



**FIGURE 7. CEO Set Optimization Sequence and Constraints Window**

Clicking on the Run button takes us to the *Examine Optimal Sequence Display* Window (Figure 8). Here we note that out of the 15,360 potential configurations for evaluating system cost versus the probability of a critical system failure, *CEO* has found 27 optimal configurations. These configurations are ordered according to increasing critical failure probability. The optimal system configuration is shown in the example. The push buttons below the *Subsystem/Configuration* list enables us to scan through the other configurations in the dominating sequence.

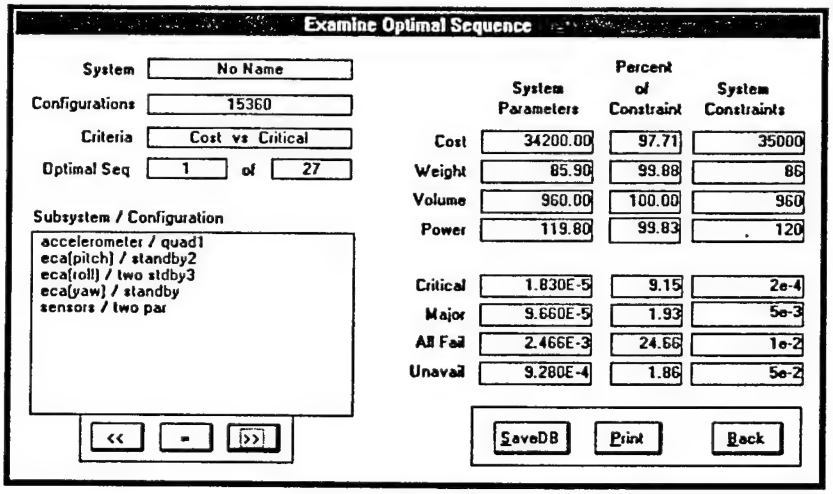



**FIGURE 8. CEO Examine Optimal Sequence Window**

The actual system parameters, as well as the system level constraints, are in the data display boxes on the right hand side of the window. The same display area also presents individual system parameter values as a percent of the corresponding system constraints.

## 5.0 References


[Aris 64] Aris, R., *Discrete Dynamic Programming*, Blaisdell Publishing Company, 1964.

[Bellman 57] Bellman, R.E., *Dynamic Programming*, Princeton University Press, 1957.

[Fan 64] Fan, L. and Wang, C., *The Discrete Maximum Principle*, John Wiley, 1964.

# DRTSS: A Simulation Framework for Complex Real-Time Systems

Matthew F. Storch and Jane W.-S. Liu
Department of Computer Science
University of Illinois
Urbana, IL 61801

**Abstract**

This paper describes an object-oriented simulation framework called DRTSS, which allows its users to easily construct discrete-event simulators of complex, multi-paradigm, distributed real-time systems. DRTSS is meant to complement complex-systems engineering tools that provide functional or object-based decomposition. Preliminary, high-level system designs can be entered into DRTSS to gain initial insight into the timing feasibility of the system. Later, detailed designs can be entered under the original design in a hierarchical fashion, and more detailed analysis can be undertaken.

DRTSS is a member of the PERTS family of timing-oriented prototyping and verification tools. In addition to DRTSS, PERTS provides a schedulability analysis tool for use with systems that allow deterministic mathematical analysis. However, several aspects of complex, real-time systems often make an *a priori* schedulability analysis difficult or impossible. In order to determine the timing properties of such systems, we turn to simulation methods. While simulation methods have been extensively used, most simulators are designed around a single scheduling paradigm and do not focus on real-time issues, and therefore are not applicable to complex real-time systems. DRTSS fills this gap by providing a suite of modern task assignment, task scheduling, and resource-access control algorithms and by allowing the rapid construction of simulators that use multiple algorithms.

## 1 Introduction

This paper describes an object-oriented simulation framework called DRTSS, which allows its users to easily construct discrete-event simulators of complex, multi-paradigm, distributed real-time systems. The primary purpose of DRTSS is to allow designers of complex real-time systems to easily enter a system design including timing parameters and to run experiments that indicate whether the system has the desired timing properties.

DRTSS is meant to complement complex-systems engineering tools that provide functional or object-based decomposition. During early stages of the design and prototyping process, high-level system designs can be entered into DRTSS to gain initial insight into the timing feasibility of the system. Later, decomposed designs can be entered under the original design in a hierarchical fashion, and more detailed analysis can be undertaken.

DRTSS is a member of the PERTS (Prototyping Environment for Real-Time Systems) family of timing-oriented prototyping and verification tools (Liu 1993). In addition to DRTSS, PERTS provides a schedulability analysis tool for use with systems that allow deterministic mathematical analysis. However, several aspects of complex real-time systems often make an *a priori* schedulability analysis difficult or impossible. For example, active resources in the system may be scheduled using different scheduling algorithms. Communication times may be nondeterministic, and so forth.

In order to determine the timing properties of such systems, we turn to simulation methods. While simulation methods have been extensively used, most simulators are designed around a single scheduling paradigm and do not focus on real-time issues, and therefore are not readily applicable to complex real-time systems. DRTSS fills this gap by providing a suite of modern task assignment, task scheduling, and resource-access control algorithms and by allowing the rapid construction of simulators that use multiple algorithms.

DRTSS differs from most other simulation frameworks in that it does not take the traditional queueing theoretical view of the target system. Instead, the target system is modelled by PERTS *task graphs* and *resource graphs*.

A DRTSS simulator is comprised of several objects. The *driver* controls the overall execution of the simulator. Each active resource has an associated *microkernal* that interfaces it to the rest of the simulator. Associated with each resource may be one or more *scheduling algorithms* that control the allocation of the resource to tasks. An *event* marks the occurrence of a scheduling-related activity. Hereafter, where there is no ambiguity, we will refer to both the DRTSS environment and a DRTSS simulator simply as DRTSS.

Following this introduction, Section 2 discusses the design objectives of DRTSS. Section 3 describes briefly the real-time systems model used by DRTSS. The scheduling algorithms supported by DRTSS and the DRTSS execution environment are described in Sections 4 and 5, respectively. Section 6 describes the DRTSS tools that support hierarchical decomposition of complex systems. The interface between DRTSS and existing data formating and performance analysis tools are described in Section 7. Section 8 describes the DRTSS capability for multiple simulation runs. Section 9 is a summary.

# 2 Goals

Our overall goal for DRTSS is to allow the user to model the widest possible variety of real-time systems in a natural manner and at the desired level of detail. We identify the following specific goals:

1. DRTSS must support the simulation of complex distributed real-time systems with many different active resources such as CPUs, networks, sensors, storage units, graphical displays, and so forth.

2. The various resources mentioned in Goal 1 are typically scheduled according to completely different algorithms. Therefore, DRTSS must be able to support many different scheduling paradigms. This minimally includes the following:

   - traditional uniprocessor and multiprocessor scheduling algorithms for general task sets;
   - scheduling algorithms for periodic tasks, with servers for aperiodic tasks;
   - low-level disk drive and network scheduling algorithms;
   - end-to-end scheduling algorithms;
   - imprecise computation scheduling algorithms; and
   - arbitrary user-defined scheduling algorithms

   In particular, DRTSS should be useful as a tool for exploring the interaction between scheduling algorithms on the various resources.

3. DRTSS should support functional and object-based decomposition methodologies, although it should not duplicate the functionality found in tools dedicated to those purposes.

4. DRTSS should allow the user to account for the overhead of scheduling algorithms and the effects of their implementations in a realistic way. A factor in scheduling overhead is the time and resources used to carry out scheduling activities. The user should be able to simulate the strategy used by a scheduler to schedule time for its own scheduling activities.

5. DRTSS must be able to actually execute tasks that have been implemented (i.e. those tasks for which the user supplies executable code). In this way, DRTSS allows the emulation of parts of the software system.

6. DRTSS must not require the user to do any (traditional) programming in order to configure and instantiate a basic simulator. Users may, at their option, implement their own scheduling algorithms and tasks as stated in Goals 2 and 5.

# 3 Real-Time Systems Model

DRTSS relies on a PERTS resource graph to describe the underlying hardware and software platform. It uses a PERTS task graph to describe the application software running on the platform in terms of the work it does. (A complete description of these models can be found in (Silberman 1994).)

Each vertex in a resource graph represents either an *active* or *passive* resource. The defining characteristic of an active resource, called a *processor* from here on, is that it can cause a task to make progress. A processor is usually referred to as a server in queueing models. Examples of processors are CPUs, disk drives, and network transmission lines. On the other hand, a passive resource alone can never cause a task to make progress. Examples of passive resources are database locks and memory. Many resources can be modelled either as active or passive depending on the point of view of the user; for instance, disk drives and network transmission lines may in many cases be modelled either way. Both processors and passive resources may have associated scheduling algorithms. Scheduling algorithms for passive resource are usually referred to as resource-access control protocols and are provided as extensions of processor scheduling algorithms. We further classify a passive resource as either a physical (hardware) or logical (software) resource.

The edges in the resource graph are of two types. A *containment edge* is directed and indicates an is-a-component-of relationship, directly supporting object-based decomposition. The subgraph containing only these edges is a forest of *component trees*. With respect to a component tree, the parent of a physical resource is fixed, but the parent of a logical resource may change at run time. In other words, a logical resource may *migrate*.

An *accessibility edge*, on the other hand, is undirected. At least one of the vertices adjacent to an accessibility edge represents a processor. A task running on a processor represented by the processor vertex adjacent to an accessibility edge can (remotely) access any passive resource represented by a vertex in the component tree rooted at the vertex at the other end of the edge.

The task graph is directed and acyclic. Each vertex represents a task to be run on some processor. Edges in the task graph represent dependency relationships between tasks. There may be more than one connected component in the graph. Tasks may be periodic, aperiodic with multiple executions, or single-execution.

# 4 Scheduling Algorithms

To meet Goal 2, we developed the notion of a "software backplane" interface, which allows all manner of scheduling algorithms to "plugged into" a DRTSS simulator. The user can add custom scheduling algorithms and resource-access control protocols that follow the protocol of the interface. However, in keeping with Goal 6, PERTS will be supplied with most common modern scheduling algorithms and resource-access control protocols, so users will not be required to implement these algorithms.

Again, associated with each processor is a microkernal which simulates the execution of tasks on the processor according to the scheduling algorithm of the processor. The interface between a scheduling algorithm and its microkernal is based entirely on event handling. The microkernal passes each event to the scheduler at the moment in simulated time that the event occurs. The scheduler determines whether or not it is "interested" in the event – if so, the scheduler preempts the task that is currently running on the processor. The scheduler may or may not require any simulated time to execute, depending on how careful the user wants to be about scheduling overheads. If the scheduler is not interested in an event, it merely ignores the event.

One benefit of implementing scheduling algorithms as event handlers is that the algorithms become modular rather than monolithic. Since the handlers are independent except for explicitly shared data structures and each is meant to handle one particular type of event, it is possible to combine handlers that provide one scheduling function (say local scheduling on one processor) with handlers that provide other functions (say task assignment or resource allocation), as long as the algorithms implemented by them are not intrinsically incompatible. (For instance, the priority ceiling protocol cannot be used with a dynamic priority scheduling algorithm.)

At the level of the software backplane, DRTSS provides a framework that scheduling algorithms can plug into, and this framework exploits what common ground there is between disparate scheduling algorithms. Algorithms for different scheduling and resource allocation functions are often not independent. For example,

to insure that all tasks assigned to each processor can be feasibly scheduled to meet their deadlines, most algorithms that assign periodic tasks to processors require the maximum processor utilization (called the schedulability bound) as input. This bound is a function of the algorithm used locally on each processor to schedule tasks on the processor. In the worst case, any peculiarities and dependencies between particular pairs of algorithms must be dealt with in the implementations of those algorithms on a case-by-case basis. However, many compatibility and interaction issues can be dealt with for entire classes of algorithms in one shot. For instance, all local scheduling algorithms with known schedulability bounds should provide a method which task assignment algorithms can invoke to determine the bound.

Determining families of interoperable algorithms is a challenging problem in its own right. The problem is made harder in that there are three levels of interoperability: a given pair of algorithms may be able to coexist on one processor, coexist on different processors, or not coexist at all. We will tackle this problem for the scheduling algorithms implemented in PERTS.

# 5 Execution of Tasks and Scheduling Algorithms

In order to meet Goals 4 and 5, DRTSS handles tasks and scheduling algorithms in a non-traditional manner. The most direct way to account for scheduling overhead and to model scheduling activities is for the simulator to "execute" scheduling algorithms the same way it executes tasks in the simulated system. Therefore, scheduling overhead can be simulated to the same degree of fidelity as any other work done by the system. This scheme assumes that when the simulator simulates the execution of each task, it actually executes the task by invoking a method of the task that contains ordinary executable code. In any case, it is necessary to assume that tasks can be "live" (that is, contain executable code) in order to meet Goal 5. Having already gone this far, we made the design decision that all tasks will be live as far as the simulator is concerned. This decision does not place any additional burden on the user. If desired, the user can create ordinary PERTS resource and task graphs, choose scheduling algorithms from those provided, and start simulating without concern for how DRTSS implements the simulation. For each task in the task graph that does not have executable code, DRTSS provides a standard live DRTSS task that takes its behavior (execution time, resource requests, etc.) from the definition of the task graph task. If on the other hand, the user wants to supply custom scheduling algorithms and executable tasks, the facilities are there for him to do so.

An additional benefit of allowing live tasks is that the user can easily implement a hybrid model as in (Chiu and Chow 1978) or (Schwetman 1978). A hybrid model allows the user to take advantage of both analytic and simulation methods. The user may have a good analytic model for certain subsystems of a large system but cannot model the system as a whole analytically. Simulation of the subsystems together with the entire system would be too costly. One way to reduce the cost of simulation is to use a hybrid model. For example, suppose that the user has a queueing theoretical formula that accurately predicts how long a task would take to execute on a complex subsystem. Then the subsystem could be modelled at the task graph level as a simple processor, and the queueing theoretical formula could be evaluated whenever the executable code of the task is invoked, in order to determine the execution time of the task.

# 6 Hierarchical Decomposition

Decomposition, either functional or object-based, is an important tool in support of the synthesis of complex real-time systems. PERTS task graphs allow tasks to be decomposed. A task graph is recursively defined in the sense that any vertex can itself be a (decomposed) task graph. When we mean to emphasize the relationship between a task and the task graph that composes it, we refer to the vertex as a supertask or the task graph defining the supertask as a subgraph. This feature serves two distinct purposes.

## Information Hiding

The first purpose is information hiding. To illustrate, we consider a flight control system (FCS) that is one component of a larger flight management system in the process of being designed. The FCS is likely to be

decomposed into many subcomponents in the final design. Early in the design and prototyping process, little specific information may be known about the FCS. The designers may be able to reason, however, that the FCS will be comprised of several periodic tasks. From past experience they may be able to establish a crude estimate of the total processor utilization of those tasks. A vertex can therefore be created in the task graph to represent the FCS, a supertask, and initial DRTSS simulation runs can be used to establish the feasibility of the flight management system.

Later in the prototyping process, specifications will become available that describe in detail the tasks that comprise the FCS. A subgraph can then be created to represent the tasks of the FCS. At the same time, the initial crude parameters of the FCS supertask can be updated to more accurately represent the tasks. When a subsequent DRTSS simulation is run, the designers now have the option of having DRTSS either expand or not expand the supertask. If the supertask is expanded, the simulator will ignore the supertask and simulate the subgraph instead, which results in a slower but more accurate simulation. When the designers turn their attention to other parts of the flight management system, however, they may want to turn off the expansion to reduce simulation time. Even if the expansion is always turned on, it is desirable to maintain the supertask/subgraph relationship when viewing the task graph.

### Scheduling Hierarchy

The second purpose of task subgraphs is to allow the simulation of a hierarchy of scheduling algorithms. Hierarchical scheduling occurs, for example, in a client-server architecture. As far as the underlying operating system is concerned, a server is an ordinary task to be allotted processor time and resources. Client tasks that request service from the server need to be "executed" by the server. ¿From the point of view of a client task, the server is a "processor" since it causes the client to make progress. The server must choose which client task to service next whenever there is more more than one request. Therefore, a scheduling algorithm must be specified for the server, and the time when a client task is scheduled to run is a function of both the system scheduler and the server scheduler. We call the composition of these schedulers a *scheduling hierarchy.*

Hierarchical scheduling has been used in periodic real-time systems that make use of periodic servers to handle aperiodic tasks. Algorithms for scheduling periodic servers include the sporadic server, described in (Sha 1989), and the deferrable server, described in (Lehoczky 1992). The server scheduling algorithm determines when the periodic server runs, and the server itself determines which of the currently outstanding aperiodic client tasks to run.

We can also consider a database example. The database server is represented by a supertask in the task graph, and the clients by its subgraph. The processor's scheduler determines when the server will run and controls the allocation of passive resources such as disk drives that the server may need. When the server runs, it does not make progress with its own computation; rather, the server makes progress with the computation of one of its client tasks.

Another well-known example where a scheduling hierarchy can occur is a system that uses a cyclic executive to implement a temporal firewall between groups of tasks. For example, a cyclic executive could be used in a flight management system to enforce temporal bounds between the FCS and other supertasks such as navigation and display. Since the FCS tasks are scheduled independently of tasks in other supertasks, the composition of the FCS scheduler and the cyclic executive scheduler is a scheduling hierarchy.

There is an important difference between the case of information hiding and the case of hierarchical scheduling. In the former case a supertask and the tasks that compose it are two views of the same entity, and DRTSS simulates either one or the other but not both. In the latter case the supertask and the tasks it serves are distinct and both must be simulated together.

## 7 Events and Event Lists

The output of a DRTSS simulator is a set of primitive events, such as "task_preempted" and "task_aborted," partially ordered according to the instants of their occurrences. A primitive event is a marker of a particular scheduling-related activity – it occupies only a time instant, rather than an interval. Primitive events are

parameterized as necessary to give complete information regarding what activity occurred; for example a task_preempted event is parameterized by the preempted and preempting tasks.

The set of primitive events can be viewed directly. However, in most cases, large volumes of low-level event information is neither required nor desired. Usually we are interested in counts of the occurrences, or probability distributions of the intervals between occurrences. We are also interested in detecting particular patterns of primitive events that have particular temporal relationships, which we call compound events. We are developing CELL, a language based on first-order logic, to facilitate the recognition of compound events and the extraction of pertinent timing information.

The output event list provides a clean interface between the simulator proper and output processing tools, such as analysis and display tools. The simulator and the event list processing tools communicate via Unix sockets. Using event list processing tools, we can graphically display the event list as a schedule, save it as a text file to examined by hand later, input it to an analysis program, etc. We plan to provide several analysis tools written in CELL. In addition, since the events are stored in the Pablo Self Defining Data Format (Aydt 1993) tools provided by or compatible with the Pablo Performance Analysis Environment (Reed 1993) can be used.

# 8 Multiple Simulation Runs

There are two reasons to make multiple simulation runs during one instance of DRTSS execution: collecting sample values and searching the parameter space. If some input parameters are random variables, then the simulation becomes nondeterministic, and the performance measures of interest are random variables as well. In this case, it is necessary to collect a sufficient number of sample values of the performance measures using identical input parameter values for each run.

Even when all input parameters are deterministic, a simulation may be repeated because the user may wish to search the parameter space of the simulated system. For example, the user may want to find the set of values for which the system meets some particular criteria (such as no task missing a deadline). To help automate this search, DRTSS allows almost any numeric task, resource, or simulation input parameter to be given in terms of a *range* of values. Specifically, a range specifies a lower bound, upper bound, step size, and nesting level. The relative nesting levels of two ranged parameters controls which of the variables will vary faster as the search space is traversed. If the search were to be implemented as a set of nested loops, the nesting level indicates the level of nesting of the loop corresponding to the parameter.

Unless instructed otherwise, DRTSS will explore the entire search space specified by the ranged input parameters. Using CELL, the user may define the conditions under which DRTSS is to terminate the search. In this case, when an event list analysis module that is monitoring the search detects the occurrence of the conditions specified by the user, it will notify DRTSS to terminate the search.

Any event list processing module that needs to control the number or length of simulation runs can do so using a control socket connection to DRTSS. However, although multiple analysis modules can be active during a DRTSS execution, only one can open the control socket.

# 9 Future Work

As of this writing, implementation of the basic simulator engine and several sample scheduling algorithms is complete. DRTSS is written in C++, runs in the Unix/X Window System environment, and has an OSF/Motif based user interface, as are common to all modules of PERTS. Hierarchical scheduling features will be implemented by October 1994, with event list processing modules to follow by early 1995. Current plans call for CELL to be implemented as an extension to C++.

In conclusion, we view DRTSS as a critical component of PERTS. The overall PERTS goal is to help practitioners take advantage of recent advances in scheduling theory in the prototyping, design, or re-design of complex real-time systems. DRTSS helps achieve that goal by providing complementary functionality to the schedulability analysis components of PERTS. It allows a wider variety of systems to be evaluated at a greater level of detail, at the cost of obtaining only probabilistic results in most cases and requiring significantly more CPU time than the time required by PERTS schedulability analyzer.

## Acknowledgement

This work was partially supported by the U.S. Navy ONR Contract No. NOOO14-J-92-1815.

## References

Aydt, R. A. 1993. "SDDF: The Pablo Self-Defining Data Format." Technical Report. Department of Computer Science, University of Illinois, Urbana, IL. (May)

Chiu, W. W. and W. M. Chow. 1978. "A Performance Model of MVS." *IBM Systems Journal* 17, no. 4: 444-462.

Lehoczky, J. P.; L. Sha; and J. K. Strosnider. 1992. "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments." In *Proceedings of the Real Time Systems Symposium* (Phoenix, AZ, Dec. 2-4). IEEE Computer Society Press, Los Alamitos, CA, 110-123.

Liu, J. W. S.; L. Redondo; Z. Deng; T.S. Tia; W.K. Shih; and R. Bettati. 1993 "PERTS: A Prototyping Environment for Real-Time Systems." In *Proceedings of IEEE 1993 Real-Time Systems Symposium* (North Carolina, December 1993). IEEE Computer Society Press, Los Alamitos, CA, 184-188.

Lehoczky, J. P.; L. Sha; and J. K. Strosnider. 1992. "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments." In *Proceedings of the Real Time Systems Symposium* (Phoenix, AZ, Dec. 2-4). IEEE Computer Society Press, Los Alamitos, CA, 110-123.

Reed, D. A.; R. D. Olson; R. A. Aydt; T. M. Madhyastha; T. Birkett; D. W. Jensen; B. A. A. Nazief; and Brian K. Totty. 1991. "Scalable Performance Environments for Parallel Systems." In *Proceedings of the Sixth Distributed Memory Computing Conference* (Apr.) IEEE Computer Society Press, Los Alamitos, CA, 562-569.

H. D. Schwetman. 1978. "Hybrid Simulation Models of Computer Systems." *Communications of the ACM* 21, no. 9 (Sept.): 718-723.

Sha, L.; B. Sprunt; and J. P. Lehoczky. 1989. "Aperiodic Task Scheduling for Hard Real-Time Systems." *The Journal of Real-Time Systems* 1: 27-60.

Silberman, A. 1994. "RTM: A Programming Environment for Real-Time Systems." Ph.D. Thesis (in preparation). Department of Computer Science, University of Illinois, Urbana, IL.

# APPLICATION OF THE ANALYTIC HIERARCHY PROCESS

# TO COMPLEX SYSTEM DESIGN EVALUATION

Michael L. Talbert
Osman Balci
Richard E. Nance

Department of Computer Science
and
Systems Research Center

Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106

## ABSTRACT

This paper examines the use of the Analytic Hierarchy Process (AHP) to weight indicators used in the evaluation of complex system designs which involve software, hardware, and humanware. Since such a comprehensive design can easily include hundreds of system quality indicators, evaluators need a technique to ensure the identification and emphasis of salient indicators in the determination of the quality of the design. The AHP is a popular technique for determining relative worth among a set of elements. In the present work, we introduce AHP with a simple example, then illustrate the application of the AHP to design evaluation using a subset of indicators from the human component of a system. We note in some detail issues which require added attention when applying the AHP to this domain. The issues include indicator selection, dealing with large numbers of indicators, incorporating group judgments, and conflict resolution. We found AHP to be an effective tool for use in assigning weights of criticality in indicator-based design evaluation, and propose elements of an environment in which the use of AHP is easily incorporated.

## 1. INTRODUCTION

A complex Navy system is composed of three major components: software, hardware, and humanware, each of which must work in synergy through complex interactions and interfaces (see Figure 1). These components are intertwined with real-time mission-critical characteristics, posing significant technical challenges for system designers, developers, and maintainers independent of warfare domain.

A critically important phase in the engineering of complex systems methodology is system design measurement and evaluation. The commitment to spend millions of dollars for building a system must be justified by thorough and in-depth evaluation of the proposed design considering the array of operational environments in which the system might be deployed.

Measurement at these scales is an important activity of interest in many disciplines. However a standard terminology does not exist. Similar terms are widely applied in various disciplines to convey the same notion. An extensive survey of the literature in the fields of hardware, software, and human factors revealed the following suite of synonymous terms: "metric," "measure," "index," "scale," "factor," "figure of merit," and "indicator." In this paper, we use the term *indicator* and define it as an indirect measure of a qualitative concept. We define *metric* as measurable component in the expression for an indicator, the value of which can be computed by a formula or measured directly.

The common goal in disciplines applying these and other related terms is, of course, to try to accurately measure a concept which can be either quantitative or qualitative. Measurement of quantitative concepts (e.g., response time, throughput, utilization) can be done directly, whereas qualitative concepts (e.g., design utility, maintainability, complexity) must be measured indirectly.

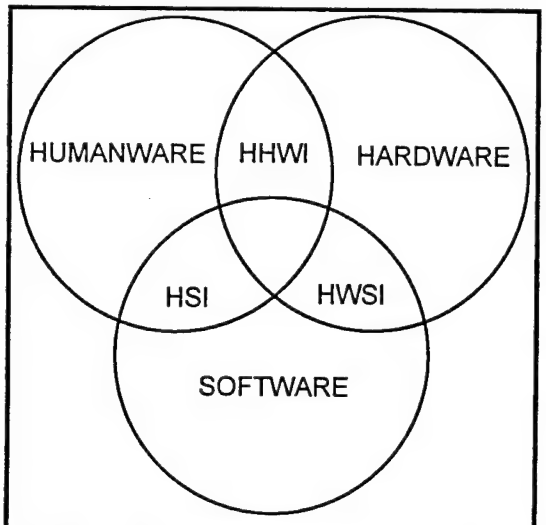


**Figure 1. Components of a Complex System.**

The most common approaches to assigning values to metrics are through direct measurement or system simulation. We can directly measure response time to a user request, but are better served by simulation in the measure of, for example, resource utilization. More subjective methods are applied in indirect measurement schemes. Balci *et al.* [1993] propose the use of the expert opinions of system design engineers and application domain specialists. These indicators can be expressed as either a single value or a range of values, referred to in the literature as "crisp" or "fuzzy" measures, respectively [Kaufmann and Gupta 1985].

A classic example structure for depicting indirect measurement of indicators is a hierarchy (Figure 2). In a hierarchical approach, concepts are measured by a set of composite indicators. These indicators themselves may be measured by composite indicators, until at the bottom of the hierarchy, we have lowest level indicators, or "leaves," in the sense of a graph. These lowest level indicators must be assigned crisp or fuzzy values.

The composite score for the top level concept is intuitively a function of the leaf scores, aggregated all the way up the hierarchy. The aggregation process involves two phases: (1) assigning or computing *weights of criticality* for all indicators from the leaves to the root, and (2) computing the overall score for the top-level concept (the design element) from the scores assigned to the leaves.

In assigning weights to individual or aggregate indicators, we attach a level of significance, representative of the contribution of the indicators to the quality of the design. The weight of an indicator is our assurance that the most important aspects of the design, in our opinion, are judged the most critically, and the less important ones less so. In so doing, we strive to ensure that the value of the design is measured not as merely the sum of the values of the components; it is more the case that the evaluation of quality of the entire design reflects the degree to which all the components are "pulling their weight."

The assigning of the weights then becomes a critical step in evaluating the design. We propose

the Analytic Hierarchy Process (AHP) as a means to assign weights to indicators

The purpose of this paper is to present how the AHP can be used for the evaluation of complex system designs by providing a mechanism for deriving weights for system indicators. Section 2 presents an overview of the AHP and illustrates its use by a simple example. Section 3 provides a more detailed example, applying the process to a suite of system-level indicators germane to the human element of a complex system. The section also addresses some special attention needs. Finally, section 4 provides a brief summary, states the conclusions of our research, and introduces directions for future research. These include (1) the application of fuzzy mathematics as the mechanism for score assignment and aggregation, since some values are derived from subjective judgment, and (2) the incorporation of knowledge-bases and rule sets as a means of ensuring consistency and validity in the weighting and scoring processes.

## 2. THE ANALYTIC HIERARCHY PROCESS: AN OVERVIEW

The Analytic Hierarchy Process [Saaty 1980] comes from the field of decision theory and has been widely applied in fields ranging from finance [Srinivasan and Bolster 1990], to software engineering [Toshtzar 1988], to combat force evaluation [Lee and Ahn 1991]. It is often used in decision making when choosing among alternatives based upon their relative worth. Its use is keyed on the methodical application of expert judgments to determine the relative contribution of factors to a decision. Saaty's structure for analyzing a decision is a hierarchy and his mathematical vehicle for computation is a positive reciprocal square matrix. Details of the mathematics are beyond the scope of this paper, and the reader is referred to Saaty's original work. AHP has been widely studied and the literature is rife with its adaptations, extensions, criticisms, and applications (see e.g., [Murphy 1993; Masuda 1990; Harker 1987a,b; Saaty 1987; DeTurck 1987; Weiss 1987; Zahedi 1986]).

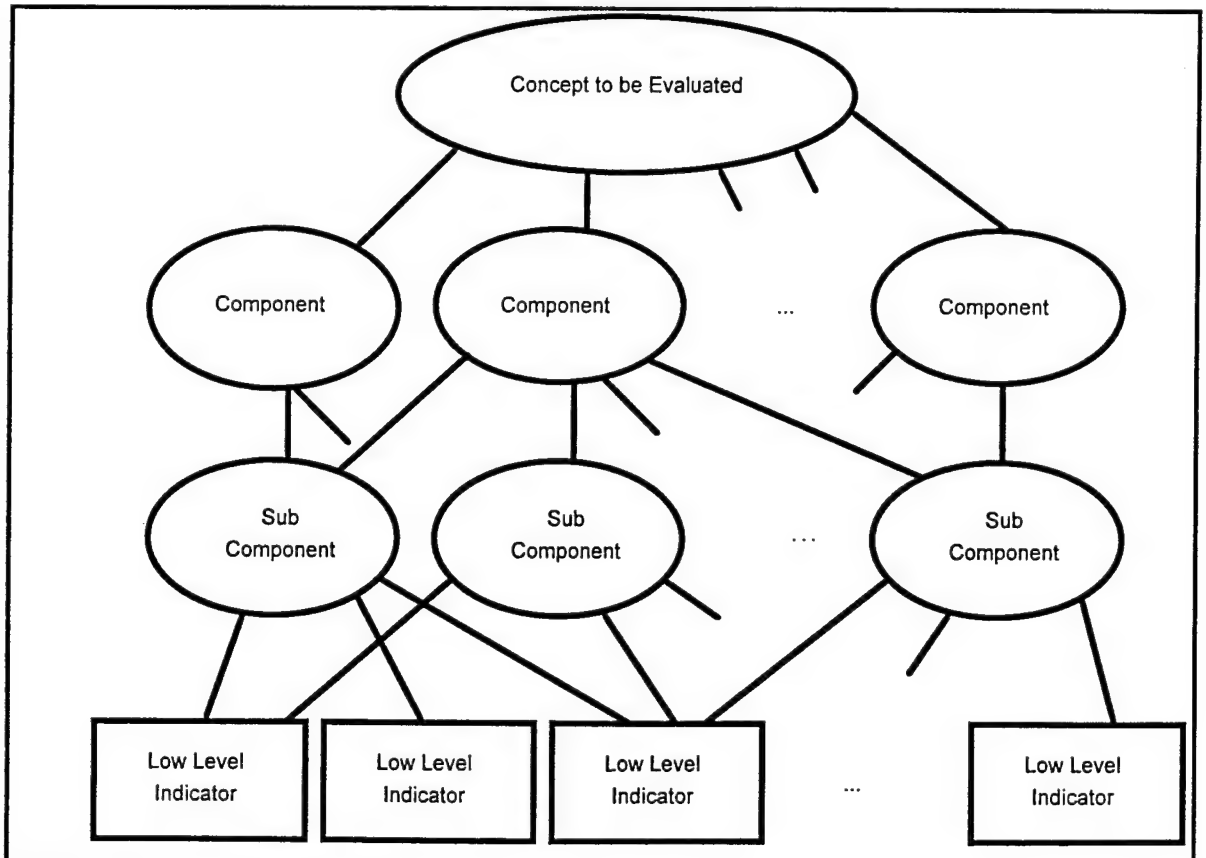



**Figure 2. Typical Evaluation Hierarchy.**

## 2.1 How AHP Works

The basic phases to applying the AHP are as follows:

1. Define the decision issue and bound its scope.
2. Structure a hierarchy of the elements contributing to a decision.
3. In a bottom-up fashion, construct a pairwise comparison matrix of the relevant contribution or impact of each element on its governing criterion in the next higher level. Saaty [1980] has developed a pairwise comparison scale for use in computing relative worth of decision elements (see Figure 3). Extensions on this basic scheme exist, including a 1-20 scale and scales which allow real numbers [Saaty 1993; Zahedi 1986].
4. Using the eigenvector method described by Saaty [1980], compute the relative contribution of each element and test the individual matrices for consistency (reflects soundness of judgment, understanding of the interdependencies of criteria, etc.).
5. Working downward through the hierarchy, use hierarchical composition to combine the weight vectors and arrive at global and local relative contributions (priorities) of each element.
6. Perform overall consistency check on the completed hierarchy of weights. Reassign relative weights contributing to a consistency ratio out of tolerance (see section 2.2).

We omit details of the mathematics and direct the reader to the references listed above, especially [Saaty 1980] for a thorough discussion. The key to using the AHP is building the hierarchy of elements and computing their relative weights. To introduce the process, we provide a simple example taken from [Saaty 1990].

## 2.2 A Simple Example of AHP

The decision to be made is: Which car among Chevrolet (C), Thunderbird (T), and Lincoln (L) to purchase based on comfort?

The decision maker makes the value judgments that the Lincoln is twice as comfortable as the Thunderbird, the Thunderbird is twice as comfortable as the Chevrolet, and demonstrating perfect consistency, the Lincoln is judged as four times as comfortable as the Chevrolet. A brief mention of the meaning of *consistency* is appropriate here (see Figure 4). The judgments as given above are perfectly consistent because their ratios demonstrate transitive equality. Examining the judgments more closely reveals that C = (1/2)T and T = (1/2)L, so by transitivity, C should equal (1/4)L, and it does. As we shall see below, a departure from perfect consistency is generally the rule rather than the exception. Note in Figure 4 the diagonal elements of "1" indicating that each car is judged to be equally as comfortable as itself, and that symmetric entries are computed as inverses.

To compute overall priorities, the matrix is normalized as follows:

1. Obtain the column totals (7, 3.5, 1.75).
2. Divide each entry by its column total.
3. Take the average value for each row.

This produces the vector of priorities for (C, T, L) as (0.14, 0.29, 0.57). An interpretation of this is that the Lincoln culled 57 percent of the comfort vote, the Thunderbird, 28 percent, and the Chevrolet, 14 percent. Note that the computations were quite simple due to the perfect consistency of the matrix. In a less consistent matrix (Figure 5), note the difference in priorities.

The column totals of (7, 5.5, 1.5) lead to a priority vector (row averages) of (0.13, 0.21, 0.66). Note that in the presence of inconsistency, the higher scores increased and the lower score

| DEGREE OF IMPORTANCE | DEFINITION | EXPLANATION |
|---|---|---|
| 1 | Equal importance | Two elements contribute equally to the property |
| 3 | Moderate importance | Experience and judgment slightly favor one element over another |
| 5 | Essential or strong importance | Experience and judgment strongly favor one element over another |
| 7 | Very strong importance | An element is strongly favored; its dominance is demonstrated in practice |
| 9 | Extreme Importance | The evidence favoring one element over another is of the highest possible order of affirmation |
| 2,4,6,8 | Intermediate values between two judgments | Compromise is needed between two adjacent judgments |
| Reciprocals | | When element i compared to j is assigned one of the above numbers, then element j compared to i is assigned its reciprocal. |

**Figure 3. AHP Scale of Judgments.**

decreased. Saaty [1980] provides an index of consistency (CI) determined as follows:

1. Multiply each entry by its priority (Figure 6).
2. Obtain a vector of new row sums = (0.41, 0.64, 2.02).
3. Divide the vector of row sums by the vector of priorities yielding: (0.41, 0.64, 2.02) / (0.13, 0.21, 0.66) = (3.15, 3.05, 3.06).
4. Compute the average of these values (approximately 3.09). It is easily verifiable that the corresponding value for a perfectly consistent matrix is 3.0. In fact, it is a convenient property of any such special $n$-by-$n$ matrix that the CI (in actuality the maximal eigenvalue) will always be the value $n$.
5. Subtract the value for the consistent matrix from that for the inconsistent matrix and dividing by $n$-1 yields (3.09 - 3)/2 = 0.045.
6. Consult a table of "standard" consistency indicies to find for a matrix where $n$=3, that CI = 0.58. These values represent CI's for matrices with random entries from the range 1/9, 1/8, ... , 8, 9. For reference, we reproduce these values in Table 1 [Saaty 90]. The reader is directed to [Murphy 1993] for a discussion of the limiting nature of the CI.
7. Divide the approximation CI's by the value obtained in Step 6, and obtain the consistency ratio (CR) 0.045/0.58 = 0.08.

Saaty [1980] posits that a CR of less than 0.10 indicates satisfactory consistency. See also [DeTurck 1987] for a discussion of this value.

In concluding this section, it is important to point out that the foregoing examples were contrived and embarrassingly trivial. Intuitively, as the problem size grows to the scale of complex systems, perfect consistency is predictably impossible in practice, due in large part to the nature of human judgment and numerous dependencies among indicators. Section 3 presents an example of a larger scale, which incorporates the steps of the procedure.

| COMFORT | | C | T | L |
|---|---|---|---|---|
| | C | 1 | 1/2 | 1/4 |
| | T | 2 | 1 | 1/2 |
| | L | 4 | 2 | 1 |

**Figure 4. Perfectly Consistent Matrix.**

| COMFORT | | C | T | L |
|---|---|---|---|---|
| | C | 1 | 1/2 | 1/4 |
| | T | 2 | 1 | 1/4 |
| | L | 4 | 4 | 1 |

**Figure 5. Less Consistent Matrix.**

| COMFORT | | C | T | L |
|---|---|---|---|---|
| | C | 0.13 | 0.11 | 0.17 |
| | T | 0.26 | 0.21 | 0.17 |
| | L | 0.52 | 0.84 | 0.66 |

**Figure 6. Normalized Judgment Matrix.**

## 3. USE OF AHP TO ASSIGN WEIGHTS TO INDICATORS

An important link between the AHP and our indicator-based method of system design evaluation is the hierarchical structure of the problem. However, instead of a hierarchy of decision factors, we have a hierarchy of indicators. Applying the same techniques as in AHP, we obtain a method for determining relative weights of indicators in the assessment of the design.

### 3.1 Example Application of AHP to Indicator Weighting

Following the method of Saaty, we will state the concept to be evaluated, construct the hierarchy, then compute weights and consistency indices. Additionally, we provide an illustrative example.

1. The concept to be evaluated is: Relative weights of the contribution of human-factors elements to design evaluation.
2. Construction of indicator hierarchy: The hierarchy in Table 2 is an abridgment of numerous indicators culled from the literature. It is representative of human-related indicators pertinent at the highest levels of an integrated system.

**Table 1. Consistency Indices for $n$-by-$n$ Matrices [Saaty 1990].**

| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| CI | 0.0 | 0.0 | 0.58 | 0.90 | 1.12 | 1.24 | 1.32 | 1.41 | 1.45 | 1.49 |

3. Beginning with the "leaf" indicators, we use the method of pairwise comparisons to construct the matrices of relative weight judgments. Using the judgment of the authors and the AHP tool *Expert Choice* [Saaty 1993], the we present the values in Figure 7 to depict the relative judgments of the highest level indicators. Similarly, we construct the matrices for the next level of indicators (Figures 8-10).
4. Compute the eigenvectors of the individual matrices to obtain the relative weights and consistency indices. Reconsider judgments which lead to intolerable consistency. For the matrices presented above, we compute the respective weights and consistency ratios by using the *Expert Choice* software product and present results in Figure 11 (a-d).
5. Working downward through the hierarchy, synthesize the individual relative weights to obtain local and global priorities of each indicator. These are included in Table 3.
6. Perform an overall consistency check of the entire hierarchy and make adjustments in relative judgments to compensate for poor consistency. This aggregate index is computed from the bottom up by multiplying the consistency index associated with a criterion by its priority, and summing across sibling criteria, etc. Given the consistency ratios associated with our complete tree, we use Saaty's techniques in *Expert Choice* and compute an overall consistency of 0.04.

We omit the matrices for the final level of indices; however, we use their values in the computation of their global weights, as well as the consistency of the entire hierarchy.

| HUMAN FACTORS | | Human | HSI | HHWI |
|---|---|---|---|---|
| | Human | 1 | 2 | 2 |
| | HSI | 1/2 | 1 | 1 |
| | HHWI | 1/2 | 1 | 1 |

**Figure 7. Top-Level Judgment Matrix.**

| Human | | Perf | Dep | Cost | Sec |
|---|---|---|---|---|---|
| | Perf | 1 | 3 | 5 | 1 |
| | Dep | 1/3 | 1 | 5 | 1 |
| | Cost | 1/5 | 1/5 | 1 | 1/4 |
| | Sec | 1 | 1 | 4 | 1 |

**Figure 8. Human Indicator Judgment Matrix.**

**Table 2. High-level Hierarchy of Indicators.**

| LEVEL 0 | LEVEL 1 | LEVEL 2 | LEVEL 3 |
|---|---|---|---|
| Human Factors | | | |
| | Human | | |
| | | Performance | |
| | | | Skill |
| | | | Ability |
| | | | Understandability |
| | | | Aiding |
| | | Dependability | |
| | | | Vigilance |
| | | | Availability |
| | | | Attitude |
| | | | Composure |
| | | | Teamwork |
| | | Cost | |
| | | | Salary |
| | | | Benefits |
| | | | Bonuses |
| | | Security | |
| | | | Clearance |
| | | | Training |
| | | | Integrity |
| | Human-Software Interface | | |
| | | Usability | |
| | | | Function |
| | | | Anthropometrics |
| | | | Interface |
| | | | Compatibility |
| | | Dependability | |
| | | | Stability |
| | | | Accuracy |
| | | | Availability |
| | | | Redundancy |
| | | Configurability | |
| | | | Facility |
| | | | Flexibility |
| | | | Time Required |
| | | User Preference | |
| | | | Aesthetics |
| | | | User Friendly |
| | | | Work Relief |
| | | | Work Added |
| | | Cost | |
| | | | Design |
| | | | Develop |
| | | | Test |
| | | | Maintain |
| | | Security | |
| | | | Physical Access |
| | | | Encryption |
| | | | Password |
| | Human-Hardware Interface | | |
| | | Usability | |
| | | | Function |
| | | | Anthropometrics |
| | | | Interface |
| | | | Compatibility |
| | | Dependability | |
| | | | Stability |
| | | | Accuracy |
| | | | Availability |
| | | | Redundancy |
| | | Configurability | |
| | | | Facility |
| | | | Flexibility |
| | | | Time Required |
| | | User Preference | |
| | | | Aesthetics |
| | | | User Friendly |
| | | | Work Relief |
| | | | Work Added |
| | | Cost | |
| | | | Design |
| | | | Develop |
| | | | Test |
| | | | Maintain |
| | | Security | |
| | | | Physical Access |
| | | | Encryption |
| | | | Password |

| HSI | | Usab | Dep | Conf | Pref | Cost | Sec |
|---|---|---|---|---|---|---|---|
| | Usab | 1 | 3 | 7 | 6 | 3 | 1/2 |
| | Dep | 1/3 | 1 | 4 | 4 | 3 | 1/4 |
| | Conf | 1/7 | 1/4 | 1 | 1 | 1/2 | 1/6 |
| | Pref | 1/6 | 1/4 | 1 | 1 | 1/2 | 1/5 |
| | Cost | 1/3 | 1/3 | 2 | 2 | 1 | 1/5 |
| | Sec | 2 | 3 | 6 | 5 | 5 | 1 |

**Figure 9. HSI Indicator Judgment Matrix.**

| HHWI | | Usab | Dep | Conf | Pref | Cost | Sec |
|---|---|---|---|---|---|---|---|
| | Usab | 1 | 3 | 5 | 2 | 2 | 1/2 |
| | Dep | 1/3 | 1 | 4 | 5 | 1/2 | 1/4 |
| | Conf | 1/5 | 1/4 | 1 | 2 | 1/3 | 1/6 |
| | Pref | 1/2 | 1/5 | 1/2 | 1 | 1/2 | 1/7 |
| | Cost | 1/2 | 2 | 3 | 2 | 1 | 1/5 |
| | Sec | 2 | 4 | 6 | 7 | 5 | 1 |

**Figure 10. HHWI Indicator Judgment Matrix.**

## 3.2 Additional Issues

As demonstrated here, the application of the basic AHP methodology is straightforward, especially with the aid of a software tool such as *Expert Choice* used here. However, there are issues inherent in the nature of complex systems which beg additional attention. Not the least of these issues is the selection and structuring of the indicators. Also, the size of an indicator hierarchy for a system with the complexity of, e.g., a naval weapon system, is expected to be very large. Additionally, there are likely to be interdependencies among system indicators. Finally, the issues of group decisions and conflict resolution are certainly to be considered. These issues have been addressed in the literature [Harker 1987a,b; Saaty 1990, 1987, 1980] and are within the capabilities of the AHP and its extensions, albeit not without added cost and complexity. We address these issues below.

### *3.2.1 Choice and Structure of Indicators*

The selection of the indicators and the construction of the hierarchy is not necessarily a simple task. It can require in-depth knowledge, at increasing degrees of granularity, of the system being designed. Such knowledge is not found in a single individual. It is likely to be found in an interdisciplinary workgroup of domain-specific experts. Depending on the system, the group could likely be comprised of software engineers, hardware engineers, and human factors specialists, as well as warfare experts.

The task of the workgroup would be to select the appropriate indicators for the particular level of abstraction at which they are evaluating the design. This could take place separately at the highest levels, intermediate levels, and so on. Saaty [1990] suggests that the AHP can even be used in the decision to keep or discard some of a collection of indicators. Individual preferences from group members would be averaged and used to weight

(a) Human Factors

| Human | HSI | HHWI | | CR |
|---|---|---|---|---|
| 0.5 | 0.25 | 0.25 | | 0.000 |

(b) Human

| Perf | Dep | Cost | Sec | | CR |
|---|---|---|---|---|---|
| 0.415 | 0.235 | 0.064 | 0.286 | | 0.060 |

(c) HSI

| Usab | Dep | Conf | Pref | Cost | Sec | | CR |
|---|---|---|---|---|---|---|---|
| 0.283 | 0152 | 0.044 | 0.047 | 0.078 | 0.395 | | 0.036 |

(d) HHWI

| Usab | Dep | Conf | Pref | Cost | Sec | | CR |
|---|---|---|---|---|---|---|---|
| 0.222 | 0.129 | 0.053 | 0.052 | 0.129 | 0.415 | | 0.083 |

**Figure 11. Indicator Weights and Consistency Ratios.**

**Table 3. Computed Priorities of All Indicators.**

| | Local Priority | Local/Global | | Local Priority | Global Priority |
|---|---|---|---|---|---|
| Human | 0.500 | | | | |
| | Performance | 0.4147 | | | |
| | | 0.2070 | Skill | 0.5480 | 0.1136 |
| | | | Ability | 0.2800 | 0.0581 |
| | | | Understandability | 0.0860 | 0.0178 |
| | | | Aiding | 0.0860 | 0.0178 |
| | Dependability | 0.2352 | | | |
| | | 0.1176 | Vigilance | 0.3300 | 0.0388 |
| | | | Availability | 0.3960 | 0.0466 |
| | | | Attitude | 0.0460 | 0.0054 |
| | | | Composure | 0.1320 | 0.0155 |
| | | | Teamwork | 0.0970 | 0.0114 |
| | Cost | 0.0638 | | | |
| | | 0.0319 | Salary | 0.7090 | 0.0226 |
| | | | Benefits | 0.1790 | 0.0057 |
| | | | Bonuses | 0.1130 | 0.0036 |
| | Security | 0.2863 | | | |
| | | 0.1432 | Clearance | 0.2380 | 0.0341 |
| | | | Training | 0.6250 | 0.0895 |
| | | | Integrity | 0.1360 | 0.0195 |
| HSI | 0.250 | | | | |
| | Usability | 0.2833 | | | |
| | | 0.0708 | Function | 0.5640 | 0.0400 |
| | | | Anthropometrics | 0.1920 | 0.016 |
| | | | Interaction | 0.1790 | 0.0127 |
| | | | Compatibility | 0.0650 | 0.0046 |
| | Dependability | 0.1518 | | | |
| | | 0.0376 | Stability | 0.1220 | 0.0046 |
| | | | Accuracy | 0.3670 | 0.0139 |
| | | | Availability | 0.3960 | 0.0150 |
| | | | Redundancy | 0.1140 | 0.0043 |
| | Configurability | 0.0442 | | | |
| | | 0.0110 | Facility | 0.4000 | 0.0044 |
| | | | Flexibility | 0.2000 | 0.0022 |
| | | | Time Required | 0.4000 | 0.0044 |
| | User Prefererence | 0.0474 | | | |
| | | 0.0118 | Aesthetics | 0.0490 | 0.0006 |
| | | | User Friendliness | 0.2050 | 0.0024 |
| | | | Work Relief | 0.3980 | 0.0047 |
| | | | Work Added | 0.3480 | 0.0041 |
| | Cost | 0.0779 | | | |
| | | 0.0195 | Design | 0.0880 | 0.0017 |
| | | | Develop | 0.2830 | 0.0055 |
| | | | Test | 0.2620 | 0.0051 |
| | | | Maintain | 0.3670 | 0.0072 |
| | Security | 0.3954 | | | |
| | | 0.0988 | Physical Access | 0.0570 | 0.0056 |
| | | | Encryption | 0.5970 | 0.0591 |
| | | | Password | 0.3460 | 0.0342 |
| HHWI | 0.250 | | | | |
| | Usability | 0.2215 | | | |
| | | 0.0554 | Function | 0.6090 | 0.0337 |
| | | | Anthropometrics | 0.1730 | 0.0096 |
| | | | Interaction | 0.0790 | 0.0044 |
| | | | Compatibility | 0.1390 | 0.0077 |
| | Dependability | 0.1293 | | | |
| | | 0.0323 | Stability | 0.2500 | 0.0081 |
| | | | Accuracy | 0.2500 | 0.0081 |
| | | | Availability | 0.2500 | 0.0081 |
| | | | Redundancy | 0.2500 | 0.0081 |
| | Configurability | 0.0530 | | | |
| | | 0.0133 | Facility | 0.6000 | 0.0080 |
| | | | Flexibility | 0.2000 | 0.0027 |
| | | | Time Required | 0.2000 | 0.0027 |
| | User Preference | 0.0520 | | | |
| | | 0.0123 | Aesthetics | 0.0550 | 0.0007 |
| | | | User Friendliness | 0.0840 | 0.0011 |
| | | | Work Relief | 0.4230 | 0.0055 |
| | | | Work Added | 0.4380 | 0.0057 |
| | Cost | 0.1295 | | | |
| | | 0.0324 | Design | 0.0880 | 0.0028 |
| | | | Develop | 0.1300 | 0.0042 |
| | | | Test | 0.2110 | 0.0068 |
| | | | Maintain | 0.5710 | 0.0186 |
| | Security | 0.4146 | | | |
| | | 0.1037 | Physical Access | 0.3330 | 0.0345 |
| | | | Encryption | 0.3330 | 0.0345 |
| | | | Password | 0.3330 | 0.0345 |

competing indicators. Those finishing in some agreed upon percentile would be deemed "keepers." The others could be discarded or used otherwise, e.g., as tie-breakers or at another level of granularity. Group judgments are discussed in [Saaty 1990; 1980] and are supported by *Expert Choice*.

### 3.2.2 *Large Numbers of Indicators*

Some simple arithmetic gives insight into the cost of using AHP when large numbers and deeply nested indicators are involved. The number of comparisons among $n$ sibling indicators is $(n)(n\text{-}1)/2$.

If we let $n$ be the average number of children per parent node and $d$ be a depth of the hierarchy (root at $d=1$), the number of pairwise comparisons required is $(n^d\text{-}n)/2$. Table 4 below reveals the fast growing nature of this value. In the table, the number of children per parent increases downward and depth in the tree increases to the right. To obtain the values in the table, we assume a complete tree of n nodes per parent at each depth. For example, a complete hierarchy with a depth of five (5) and an average of five (5) child indicators per parent indicator would require at worst case 1560 pairwise comparisons.

Fortunately, the worst case need not always prevail. In many example hierarchies provided in [Saaty 1990, 1980] and elsewhere in the literature, it is likely that the hierarchy is not necessarily full, reducing the number of required comparisons. Regardless, when the number of comparisons to be made becomes larger than would appear manageable, alternative means for completing massive comparison matrices or matrices with missing judgments have been derived [Harker 1987a]. Additional alternatives are illustrated in [Saaty 1990], including dividing judgments among a panel of decision makers and then combining them to obtain a complete evaluation matrix.

### 3.2.3 *Interdependencies Among Indicators*

Saaty [1990, 1987] discusses two types of interdependencies among hierarchical components -- additive and synergistic. In the *additive* case, contributions of interdependent components are simply summed. In the *synergistic* case, "the impact of the interaction of the elements is greater than the sum of the impacts of the elements, with due consideration given to their overlaps" [Saaty 1990, p. 90]. Accounting for such interdependence begins with creating new entities in the hierarchy for each interaction.

In the case of a design involving humanware, hardware, and software, such interactions abound. We have represented these interactions in Figure 1 by new entities for the human-software interface (HSI), human-hardware interface (HHWI), and hardware-software interface (HWSI). We employ the AHP techniques to judge the degree to which each interaction element contributes to the whole. An additional entity can be created for the hyper-interaction of these six entities. This region of interaction is of particular interest at the highest level view of system design evaluation. For our purposes, obtaining its measure of predicted success is an indirect indicator of the quality of the entire design.

### 3.2.4 *Group Judgments and Conflict Resolution*

The use of working groups in employing the AHP serves manifold purposes. Not the least of these is the opportunity to avail decision makers of alternative perspectives and possibilities, which may shed new light on the decision making process. Another key benefit is the opportunity to divide the work, reducing any attendant frustration associated with the application of AHP to a very large problem -- especially if the process is unfamiliar to the decision-making body. Still another benefit is the opportunity to mitigate the impact of extreme opinions based on biases, agendas, power struggles, etc. [Saaty 1990].

When conflicts do arise regarding the assignment of a pairwise comparison, and the consequences are significant, the AHP itself can be applied to a resolution. The simplest method is to employ an averaging algorithm to obtain a mean of

**Table 4. Number of Pairwise Comparisons.**

| Nodes/Depth | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 3 | 3 | 12 | 39 | 120 | 363 | 1092 |
| 4 | 6 | 30 | 126 | 510 | 2046 | 8190 |
| 5 | 10 | 60 | 310 | 1560 | 7810 | 39060 |
| 6 | 15 | 105 | 645 | 3885 | 23325 | 139965 |
| 7 | 21 | 168 | 1197 | 8400 | 58821 | 411768 |
| 8 | 28 | 252 | 2044 | 16380 | 131068 | 1048572 |
| 9 | 36 | 360 | 3276 | 29520 | 265716 | 2391480 |
| 10 | 45 | 495 | 4995 | 49995 | 499995 | 4999995 |

the competing judgments. However, with most averaging techniques, the true judgments are no longer represented in the outcome.

Another method is the application of the AHP to the determination of the relative worth of each judge's input. This technique should be applied with prudence, and is most effective when all parties are objective and agreeable to the process. The usual influences of power, politics, and position must be mitigated as much as possible for the technique to have valid results. Here is the basic process.

1. Set as a goal the assignment of an acceptable judgment to the element(s) in question.
2. Construct a hierarchy (as abbreviated as possible) of the areas of expertise which are required to arrive at such a judgment.
3. Using crisp metrics as a basis for objectivity, (e.g., "5" versus "several"), make pairwise comparisons between the judges as they apply to each of the criterion in the hierarchy from Step 2.
4. Complete the matrices, compute the weights, and determine the consistency in the usual way.
5. When the matrices achieve consistency, the result should reveal the relative worth of each judge's input. Again, with prudence as a guide, this ordering can be applied to choose the value to assign to the judgment.

## 4. CONCLUDING REMARKS AND FUTURE RESEARCH

We have defined the problem under study as the need for an appropriate scheme to provide relative worth to indicators used in the evaluation of a complex system design. We propose the use of the Analytic Hierarchy Process as a suitable numerical technique and demonstrated its use by examples. We note and address the issues of problem size, problem structure, and conflict resolution when applying group judgments.

Considering the examples and discussion of additional issues presented here, we can conclude that the Analytical Hierarchy Process is a suitable vehicle for assigning weights to indicators as we have done. Though application of the process is not without what can be considerable effort, (especially for the evaluation of systems with large numbers of indicators), the incorporation of software aids such as *Expert Choice* eases the burden on the evaluators.

Regarding future research, we direct our attention to two primary fronts. First, in the vein of Paek, *et al.* [1992], we propose the application of fuzzy mathematics in the assignment and aggregation of indicator scores. This technique allows for the score assignment of either a range of quantitative values, or a qualitative value which can be translated into a fuzzy numerical value. The computation of a final "score" for the design will then be the fuzzy mathematical combination of the indicator scores, weighted by their relative worth as determined by application of the AHP. The ultimate judgment of the quality of the design can then be determined by the degree to which the aggregate score satisfies the design's governing requirements specification. Future research will focus on greater detail of the application of this technique and the derivation of the range of values applicable to a requirements specification.

Secondly, we propose the incorporation of a knowledge-based environment to assist the evaluators in both assigning weight judgments and to achieving greater consistency in the overall assignments. While *Expert Choice* eases the construction of judgment matrices and the computation of the weights and consistency values, there remains room for incorporation of knowledge from the experience of system designers and domain experts to more precisely attach relative judgments. We submit that an appropriate rule base can provide several types of assistance toward achieving this end. Rules can warn evaluators of judgments which differ significantly from those of other evaluators. Additionally a suitable rule base could be used to identify judgments which can lead to inconsistency. Finally, we foresee the use of a knowledge base to serve as an advisor to design evaluators, by providing access to inputs and comments appropriate to the type of system under evaluation, based on previous systems as well as the inputs of previous evaluators.

## REFERENCES


Balci, O., D. DeVaux and R. Nance (1993), "Measurement and Evaluation of Complex Navy System Designs," In *Proceedings of the Complex Systems Engineering Synthesis and Assessment Technology Workshop (CSESAW '93)*, S.L. Howell and W. Farr (Eds.), Naval Surface Warfare Center Dahlgren Division, Dahlgren, VA, 126-140.

DeTurck, D.M. (1987), "An Approach to Consistency in the Analytic Hierarchy Process," *Mathematical Modelling 9*, 3-5, 345-352.

Harker, P. (1987a), "Alternative Modes of Questioning in the Analytic Hierarchy Process," *Mathematical Modelling 9*, 3-5, 353-360.

Harker, P. (1987b), "Incomplete Pairwise Comparisons in the Analytic Hierarchy Process," *Mathematical Modelling 9*, 11, 837-848.

Kaufmann, A. and M. Gupta (1985), *Introduction to Fuzzy Arithmetic: Theory and Applications*, Van Nostrand Reinhold Co., New York.

Lee, Y.W. and B.H. Ahn (1991), "Static Valuation of Combat Force Potential by the Analytic Hierarchy Process," *IEEE Transactions on Engineering Management 38*, 3, 237-244.

Masuda, T. (1990), "Hierarchical Sensitivity Analysis of Priority Used in Analytic Hierarchy Process," *International Journal of Systems Science 21*, 2, 415-427.

Murphy, C.K. (1993), "Limits on the Analytic Hierarchy Process from its Consistency Index," *European Journal of Operational Research 65*, 1, 138-139.

Paek, J.H., Y.W. Lee and T.R. Napier (1992), "Selection of Design/Build Proposal Using Fuzzy-Logic System," *Journal of Construction Engineering and Management 118*, 2, 303-317.

Saaty, J. (1993), *Expert Choice User Manual*, Expert Choice, Inc., McLean, VA.

Saaty, T.L. (1980), *The Analytic Hierarchy Process*, McGraw-Hill, Inc.

Saaty, T.L. (1987), "How to Handle Dependence with the Analytic Hierarchy Process," *Mathematical Modelling 9*, 3-5, 369-376.

Saaty, T.L. (1990), *Decision Making for Leaders*, RWS Publications, Pittsburgh, PA.

Srinivasan, V. and P.J. Bolster (1990), "Industrial Bond Rating Model Based on the Analytic Hierarchy Process," *European Journal of Operational Research 48*, 1, 105-119.

Toshtzar, M. (1988), "Multi-criteria Decision Making Approach to Computer Software Evaluation: Application of the Analytic Hierarchy Process," *Mathematical and Computer Modelling 11*, 276-281.

Weiss, E.N. (1987), "Using the Analytical Hierarchy Process in a Dynamic Environment," *Mathematical Modelling 9*, 3-5, 211-216.

Zahedi, F. (1986), "The Analytical Hierarchy Process: A Survey of the Method and its Applications," *Interfaces 16*, 4, 96-108.


## AUTHOR INFORMATION


***Michael Talbert*** (Capt, USAF) is a doctoral candidate researching a framework for complex system design evaluation in the Dept. of Computer Science at Virginia Tech. *(talbertm@csgrad.cs.vt.edu)*

***Dr. Osman Balci*** is an associate professor of computer science at Virginia Tech, and principle investigator of the Navy-funded research project "Measurement and Evaluation of Complex Navy System Designs." *(balci@vt.edu)*

***Dr. Richard E. Nance*** is professor of computer science and Director of the Systems Research Center at Virginia Tech. *(nance@vtopus.cs.vt.edu)*

# MODELING COMPUTER ARCHITECTURE FOR SURFACE SHIP COMBAT SYSTEMS

Attn: Code N 24 (Richard A. Holden)
Commander
Dahlgren Division
Naval Surface Warfare Center
17320 Dahlgren Road
Dahlgren, Virginia 22448-5100
(703) 663-1270

Andrew Mittura, Christopher J. Martin, Lori D. Payne
SYSCON Corporation
Dahlgren, Virginia 22448

Mitchel S. Karp
K&K Software Engineering, Incorporated
Flint Hill, Virginia 22627

Robert W. Thomas
EG&G
Dahlgren, Virginia 22448

Amanda J. Harden
SIMMS Industries, Incorporated
Dahlgren, Virginia 22448

Send all correspondence to Richard A. Holden.

Abstract

This paper presents the current approach to ongoing measurement and analysis in support of development of an AEGIS combat system model. The evolution of combat systems is reviewed in terms of the underlying computer system. The AEGIS Baseline 4 combat system is discussed in some detail and a description of the AN/UYK-43 computer system is given in terms of the theory and analysis of the design implementation. Measurement of the AN/UYK-43 system is described in terms of experimental procedures and data collection methods. Examples of measurement results are presented in terms of central processing unit response to tactical load. Results of queuing theory analysis of the AN/UYK-43 data as a M/G/1 (*M*arkov input, *G*eneral service time distribution, single server) system indicates that the system is not Markovian. Additional conclusions germane to combat system design synthesis are drawn.

## Introduction

U.S. Naval combat systems are large and complex and must execute time-critical actions in real time. The central task of the combat system is to respond to tactical events by integration of sensor and other data so that weapons can be paired to targets in real time. Combat systems are traditionally used over a long period of time (years) and evolve slowly as new components are introduced. Evolutionary changes must be carefully managed because of the size and complexity of the system and the fact that even small changes may dramatically, even adversely, affect performance. A combat system model is needed to relate characteristics of the computer system to tactical functions. The impact of changes to the combat system can be predicted by defining the relationship between tactical actions and the underlying computer architecture.

This paper presents current work in support of the development of an AEGIS combat system model. The AEGIS combat system, shown in Figure 1, is an array of individual subsystems loosely coupled through the AEGIS weapons system (AWS) Mk-7. The Mk-7 AWS is an integrated multi-element distributed system providing overall command and control to the combat system. The combat system, which includes Mk-7 AWS, is designed along the classic combat system functions of detect, control, and engage. Understanding this architecture and how it responds to a tactical stimulus is basic to model development.

The following sections provide a brief overview of combat systems evolution as it relates to the computer system. A description of the computer system is given in terms of modeling and architecture of the implementation. Next, a detailed discussion of experimental procedures and data collection is provided, followed by a statistical analysis of some of the available data.

## Combat System Evolution

The upgrade of the AEGIS combat system is depicted in Figure 2, which shows evolution of the various subsystems making up the combat system. Although not explicitly shown in the figure, computer programs have evolved in concert with subsystem equipment changes. Computer program changes and computer equipment upgrades have typically been designed to allow a maximum amount of software to be reused; i.e., evolution has depended on legacy code.

The evolution of combat systems has been based on an underlying computer system architecture that uses point-to-point connection of *standard* Navy computers This approach was established by the Navy Tactical Data System during the 1950's, and in the 1960's it was adopted by AEGIS and applied with great success. The standard changed dramatically in both hardware and software characteristics, although its philosophy of use has remained practically unchanged.

As shown in Figure 3, the standard computer evolved with increasing need for speed and memory. Also seen is the increasing need for input/output (I/O) channels as the combat system grew in complexity with the addition of more subsystems. Some subsystems require unique protocols and some have special timing requirements. To meet these needs, the AN/UYK-43 was developed with two independently programmable I/O controllers (IOC) with a strong instruction set and 64 channels that can run asynchronously and support different interface types. The importance of legacy code in system evolution is evident since the AN/UYK-43 has a complete emulation capability of its predecessor the AN/UYK-7. Legacy code is also manifest in the evolution of computer language in that CS-1 is a subset of CMS-2.

In recent years, there have been significant changes in computer technology. Concurrently, there is determined interest in using commercial off-the-shelf technology in place of the Navy standard computers to take advantage of lower cost and rapidly improving performance. These events have led to proposed combat system architectures incorporating many computers all interconnected by various methods such as busses and local area networks (LAN), or by adding adjunct processors into current computer systems. These architectures proffer advantages in reliability, survivability, application of modern computer technology, and the potential to make combat systems more amenable to future upgrades.

Partitioning functions and allocating resources have proven difficult in combat systems because almost all tasks have some affect on the data that are used to execute the tasks. System performance is critically dependent on how these data and time interdependent tasks communicate and how all have access to shared information. The definition of this problem was addressed in a previous study that concluded that combat system connectivity is a direct function of the tactical requirements and once the combat system has been functionally partitioned and resources have been allocated, the performance of the system is determined.[1] The potential for new combat system architectures, which are not based on the standard computer, has created a new need to understand connectivity. The current and proposed future architecture is illustrated in Figure 4, where the connectivity of current systems is clearly evident, but for potential future systems, it becomes obscured by the connecting medium

## Modeling and Architecture

A combat system can be thought of in terms of both an implementation model and an application model. The implementation model represents the physical system as built and consists of two parts; the software and the hardware. The application model represents the intended use of the system and consists of functions. To fully understand the combat system, it is necessary to relate implementation parameters to application functions. Measurements of system performance can only be made in the implementation domain and must be mapped to the functions in the application domain.

Development of a combat system model requires experimental data to determine how tactical requirements are partitioned into the software architecture for an actual combat system. The Baseline 4 AEGIS combat system (see Figure 2) was chosen for measurement since it is a recent version that is operational at sea and available for experimental measurement. This baseline is distributed into four major tactical elements as shown in Figure 5. Each tactical element is hosted in an AN/UYK-43 computer that has two central processing units (CPU) and 64 I/O channels available for point-to-point interconnections of computer elements and other equipment. All elements use the AEGIS Tactical Executive System (ATES/43) for control of the system and for message transmission. ATES/43 interacts with tactical computer program entities defined as task modules.

ATES/43 has been designed to provide a wide range of control functions so that each element computer program can be constructed to satisfy tactical function requirements. Each of the tactical elements (Command and Decision, SPY, Weapons Control System, and AEGIS Display System (ADS)) use these control functions differently, which leads to a unique computer program architecture for each element. Experimental work is based on a method for trapping ATES/43 software trace events that reflect the interrupt and control status of the computer. Trace events can be extracted using software patches and data collection equipment that does not unduly affect the AN/UYK-43 CPU timing. Examples of trace events are listed in Table 1.

Tactical program task modules are assigned CPU service based on a scheme that uses 4 levels of preemption and 16 levels of priority. Task modules can be scheduled three different ways; *a priori* from a periodic table, initiated by another task module, or initiated as a result of an operator entry from a console. Use of the CPU is under control of the task modules except for limited system control functions such as hardware and software faults and temporary storage allocation. The task modules completely control the flow of the processing and the order in which they will be processed. ATES/43 implements the mechanics of executing modules, but does not control the use of the CPUs as is often done in operating systems.

Table 1. Examples of Trace Events.

| Hardware Interrupts |
|---|
| Hardware faults |
| Software faults |
| Task module scheduling |
| **Software Interrupts** |
| Task module service requests |
| Intracomputer messages |
| Intercomputer messages |

To characterize the system, measurements are needed in parameters specific to computer resource activity such as processing time and memory used. To map these measurements to the application, analysis of the functions must be performed to identify what external input excites a function. This functional excitation is defined in terms of tactical load; i.e., the load caused by external inputs to the system. The key then is to develop a parametric relationship between the tactical load and the computer resource activity. This relationship is termed the system response.

A response curve can be drawn for any system. For a combat system, the response curve is given as the tactical load versus CPU use. The response curve will be made up of two components; the static component, which has no correlation to the load; and the dynamic component, which has a mathematical relationship to the load. Analysis of the AEGIS combat system suggests that the response of each element is different.[2] The predicted response curve for each element of the system is shown notionally in Figure 6.

To support this experimental study, an architectural analysis was conducted on each element.[3] Since the element software was developed under MILSTD 1679, there is a program design specification (PDS) for each element. The PDSs give detailed overviews of the software architecture, describe all interfaces with ATES/43, specify all external hardware interfaces, and include high-level descriptions of all the modules. From the paper analysis, highlights of the fundamental design of the software architecture of each element has been

produced, including estimates of CPU loads for worst-case timing environments. In the process of developing the worst-case timing environments, CPU-load sensitivities to tactical requirements were projected. Diagrams were produced depicting each element's module flow for processing tactical requirements that are CPU load sensitive. The cost of each element's I/O burden, including encoding and decoding, on the CPU has been calculated. Finally, all the module entrances that were significant from the standpoint of CPU load were summarized. This information was used as a guide to define experiments and evaluate experimental data.

Timing estimates given in the PDSs for the elements yielded an interesting characteristic of CPU load. The duration of time over which the CPU response was calculated greatly influenced the resulting average, even though the system input load was held constant. The difference between a 10- or 20-sec duration could affect the resulting calculated average CPU response by as much as 20 percent. Analysis of this characteristic indicated that each element should have a natural timing cycle such that the CPU response should be constant when calculated over integer multiples of the natural timing cycle, assuming that the system input was held constant over the measurement interval.

The initial experimental emphasis has been on running scenarios that place the greatest timing load on the system. The simulator scenarios have been designed to build up the load systematically so that response curves can be established. In addition to the timing parameters of each task module, all transactions of the scheduling and dispatching queue are collected. Figure 7 illustrates these events, which include when a module first enters the queue, what action precipitated the scheduling, and when the module is dispatched.

In addition, experiments are being designed to determine the response time of the system. The response of a single element to an external stimulus can be readily measured; however, most system functions involve multi-element response since the effect of an external stimulus often migrate through all the elements before tactical action is taken. Multi-element response is difficult to measure and interpret since it involves time delays and reaction to feedback. Tactical functions supported by multiple elements are critically important since system response times could be altered with repartitioning. To determine when certain information arrives at an element, a method described in the next section is used to monitor I/O channels.

## Experimentation Procedures and Data Collection

The hardware and software configuration is critical to successful experimentation. It is desirable to have as much of the complete system as possible. The nature of combat systems however limits the amount of real system components available. For example, the facility currently being used does not have gun mounts or missile launchers. To compensate for this lack of equipment, the experimental system is wrapped in simulators. The fidelity of the simulations influence the performance and the ability to exercise certain functions. Several different input sources have been used to stimulate different functions of the combat system. Figure 8 identifies the configuration showing both the real components and the simulated components.

Repeatability is crucial in system timing measurements. This repeatability applies to the simulator scenarios, system configuration, data recording instrumentation, and reduction of collected data. Lack of repeatable scenarios limits the ability to ensure that one is not measuring characteristics of the simulator. A nonrepeatable test system configuration may lead to incorrect conclusions since the configuration influences the processing resources. Repeatability in instrumentation is necessary if the correct data are to be taken at the correct location. Reduction of data must be repeatable to ensure correct and consistent analysis and conclusions.

Current system measurements follow a rigid procedure. A test plan is developed for consistency in data collection and procedures. The configuration is verified prior to data collection to ensure all connections are made and the system is operating correctly. The scenario and the data collection are started on direction from the test director. Each scenario has a delay of 2 min to allow for data collection in the no-load condition; i.e., the computer system is idle. The entire system is reinitialized before each test so that there are no residuals from the previous test.

Two different types of data are collected—tactical events and computer events. Tactical events include system time and object data such as number and type of targets tracked and number of engagements. Examples of computer events are interrupts (hardware and software), scheduling queue entries, control transitions between executive (ATES/43) and task state code, and the number of software instructions. Computer events are then correlated with tactical events to determine the system response.

Tactical events are recorded by the AEGIS combat system computer programs that were designed to record tactical data and other parameters on magnetic tape. This collection capability is called AEGIS data recording and is done by predefined extraction points in the software. The primary purpose of data recording is to provide the necessary stimuli to recreate either a

problem experienced by the crew aboard ship or to reconstruct a key event for further analysis. Because of this intended use, data recording is done in a background mode, which has two drawbacks. First, it may not collect all available data, particularly in high-load scenarios. Second, data recording contributes to the CPU load and in many cases its affect cannot be neglected.

Computer events are collected using the AN/UYK-43 performance monitoring interface (PMI), which is a set of interconnect pins that replicate the computer backplane. The PMI has special cards that latch the various data busses and present the data in a form that is collectible with a logic analyzer. The data types available from the PMI are shown in Table 2. Each data bus is 32 bits wide with one or two control signals to indicate valid data. A data word is valid at the PMI output for 37.5 nsec, which requires a high-speed device capable of latching and recording within this interval. The rate that valid data words appear at the PMI varies depending on which bus is viewed. Trace events occur at the kilohertz rate but in the case of software instructions, the rate may exceed a megahertz. A data collection device built around a Tektronix logic analyzer has been developed under the acronym RTADS (Real Time Analysis and Debug system).

The current work involves collection of trace events, which are occurrences of control transitions in the executive code (ATES/43). Through the use of the performance monitor memory (PMM) instruction and ATES/43 patches, each event is captured over the PMI micro data bus. These data indicate what computer event is being executed but do not provide any information on the tactical load. Tactical data are captured using AEGIS data recording. To correlate these two pieces of data, one PMM instruction reads the AEGIS system time and outputs it to RTADS. This AEGIS system time is the common link between the tactical data and the computer timing data.

To ensure that the CPU response is being measured in a steady-state region, the current simulator scenarios are designed to increment the tactical load in 10 equal steps with a 120-sec wait between each step. This allows approximately 100 sec of constant load at each step. Since data are collected continuously, the data reduction process must remove the periods of transition from one step to the next. This is done by identifying when the tracks enter the system using AEGIS data recording and AEGIS data reduction and then defining steady-state windows for RTADS-collected trace event data to be processed.

The amount of data collected is significant. AEGIS data recording requires five nine-track reel tapes per 20-min scenario. RTADS records and saves 20 Mbytes of data per element for a total of 80 Mbytes of data per 20-min scenario. All RTADS data are stored and archived on 150-Mbyte Bernoulli disks.

Table 2. AN/UYK-43 PMI Data Types.

| CPU |
|---|
| Instruction physical address |
| Instruction virtual address |
| Instruction data bits |
| Operand physical address |
| Operand virtual address |
| Read operand data |
| Write operand data |
| Active status register |
| Branch physical address |
| Branch virtual address |
| Interrupt status code |
| Performance monitoring memory |
| Performance monitoring register |
| **IOC** |
| Instruction physical address |
| Instruction virtual address |
| Instruction data bits |
| Output location physical address |
| Output location virtual address |
| Input location physical address |
| Input location virtual address |
| Input data |
| Output data |
| Control memory physical address |
| Control memory virtual address |
| Control memory contents |
| Real-time clock contents |
| IOC performance monitoring |

The verification and validation (V&V) of the measurement process is critical to the success of any experiment. This process is currently undergoing V&V. There are several aspects to the validation. Test modules that run under ATES/43 are being developed to generate all the trace events at specific times. First, by knowing the order and time the events should occur, predictions can be made as to what the trace event data should look like. Second, the input data from the simulators into the system are being collected by monitoring the computer I/O. Through comparison of the input data to scenario output data, it is possible to validate the input into the system. Third, a method has been devised for counting the instructions being executed between each trace event. This capability will

be verified by counting the instructions in the test program since the number of instructions between two given trace events is known.

## Statistical Analysis

The primary objective of statistical analysis is to aid in the identification of relationships that can be used to model system functionality and performance. An ancillary benefit is that the data volume can be significantly reduced. The key products are means, variances, and distributions of statistically variable parameters such as CPU use and queuing durations over a range of tactical load conditions.

Figure 9 shows the distribution of trace events that were measured for a situation with a track load of 18 percent. The events were as defined in Figure 7, with the exception of event 4, which was not sampled in this particular experiment. Graphs of this type are helpful in qualifying data sets in terms of CPU and I/O interrupts (events 2 and 3, respectively) and quantifying activity relative to the frequency of module initiations (event 8), module exits (event 9), and queue entries (event 41).

Response of the CPU to tactical load can be determined from the time spent in idle. Figure 10 presents the average CPU response for the ADS element for a case where the response was essentially linear with track load. This result was also applicable to all individual task modules that showed significant response. For a given scenario, modules may be either static (load independent) or dynamic (load dependent). Figure 11 ranks the ADS task module sensitivities calculated as percent increase in CPU activity attributable to the module per percent increase in the track load.

Data from initial experiments was used to determine CPU idle time duration as a function of time. Duration data was statistically analyzed for the ADS computer for various track loads. The duration distributions exhibited exponential characteristics suggesting that CPU activity represented a random behavior. This early result led to additional statistical studies using queuing theory and data from later experiments.

In the notation of queuing theory, the queuing problem for the AN/UYK-43 was initially analyzed as an M/G/1 system, which indicates *M*arkov input, *G*eneral service time distribution, and a single server. For the M/G/1 system, the solution for the mean time spent in the queue, $W_Q$, is given by the Pollaczek-Khintchine formula,[4]

$$W_Q = \frac{\lambda E(S^2)}{2(1 - \lambda E(S))}$$

where $S$ is the service time requirement for the queued element, $\lambda$ is the mean rate of arrival of customers on the queue, and $E$ designates the expected value.

When this formula is applied to estimate the mean time that modules are queued, the results tend to be up to 100 times greater than the observed queue durations determined from the queue related events. There are two reasons for this.

- The arrival times of modules requesting service are not, in general, Markovian.
- Service to a module can be interrupted for other services and/or by time-slicing. This implies that the queuing phenomenon can continue even after service has been initiated on a module.

To describe the latter effect, the waiting time is defined to be the interval between the entrance of a module and its exit when the module is <u>not</u> receiving task-state service. During this interval, the CPUs may be servicing I/O or CPU interrupts, or be in task-state for another module.

Sample results illustrated in Figure 12 show measurements associated with the throughput of a specific module at a track load that was 36 percent of the nominal system capacity. The results were all reported upon completion of each individual entrance of the module. Figure 12 compares the total queue time (seconds), the total CPU time (seconds), the occupation time (seconds); i.e., the time between the beginning and ending of execution of the module, and the average CPU loading since the previous reporting time; i.e. the fraction of the time that the CPU was not idle. It can be seen that there is a strong correlation between the queue time, the occupation time, and the system use, but the queue time is generally much less significant than the occupation time.

A more complete study to predict the tactical performance of multiple networked CPUs will require the treatment of the *M/G/k* (Markov/general/multiserver) system, or even the *G/G/k* (general/general/multiserver) system since more than one server is involved, and the requests for service tend to be highly structured in time rather than random. Predictions for these conditions can only be made by simulations based on the measured statistics of event histories.

## Conclusions

The ultimate goal of the analytic and experimental work is to establish a model of surface ship combat systems. This model will relate tactical functions and computer resources as demonstrated by the track load versus response curves for single system elements. The method for development of multi-element response curves is underway. Although much work remains to be done, current results indicate that the goal is technically realizable; however, the extent and limitations of a combat system model cannot at present be defined.

Some interim conclusions germane to combat system design synthesis follow.

- *The system is task driven.* The tactical executive controls the system in the sense that it performs management (temporary memory, queues, interrupts, faults), but it does not explicitly control CPU usage. The flow of processing in the CPU is controlled by task modules as determined by tactical function requirements that set the priority of service requests.

- *Executive service is load dependent.* Current data indicate that CPU activity for the tactical executive equals less than 10 percent of that of the task modules. Tactical executive activity increases as the load track increases.

- *The CPU activity presents deterministic behavior.* Queuing statistics of current measured data show that the arrivals for service do not occur in a Markovian fashion. This can be explained in terms of the highly periodic structure of the modules. Further, the service waiting times that occur after module computations have been initiated tend to mask the true extent of the queuing process.

- *Partitioning is constrained by the resources.* The fundamental computer system architecture through hardware and software partitioning defines the connectivity options of the combat system. Partitioning is also constrained by limitations in the tactical computer for features other than CPU time; such as, addressing span, number of registers, interrupt scheme, and memory contention. Additional constraints can be imposed by the compilation system, the build system, and the control programs (ATES/43).

## References


[1] Holden, R. A. and Harrison, R. D., *Combat System Connectivity,* NSWCDD/TR-91/283, Aug 1991, Naval Surface Warfare Center, Dahlgren Division, Dahlgren, VA.

[2] *AEGIS Weapon System CPU Timing Measurement Master Plan,* SYSCON Corporation, Jul 1991.

[3] *Tactical Computer Code Module Timing Analysis; C&D, ADS, ASWCS, WCS, SPY(U),* NSWCDD Contract N60921-C-91-A205/316 CDRL, 30 Jul 1993.

[4] Morse, P. M., *Publications in Operations Research No. 1,* John Wiley and Sons, Inc., 1958.

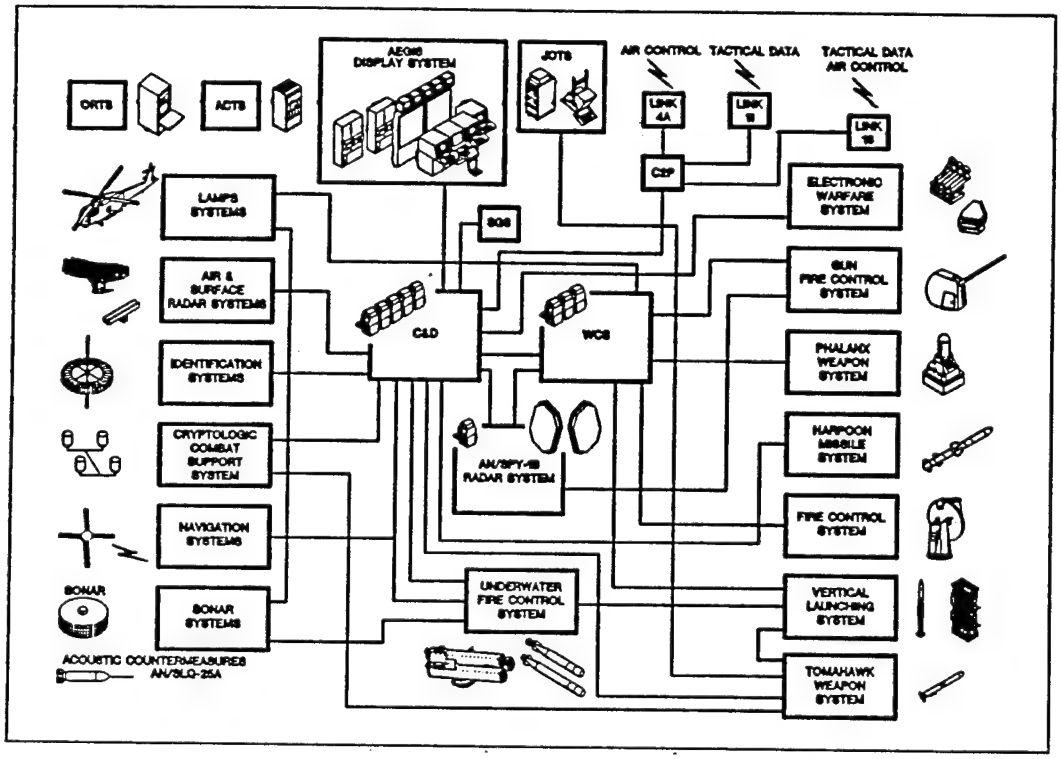



Fig. 1. AEGIS Combat System Pictorial Diagram.

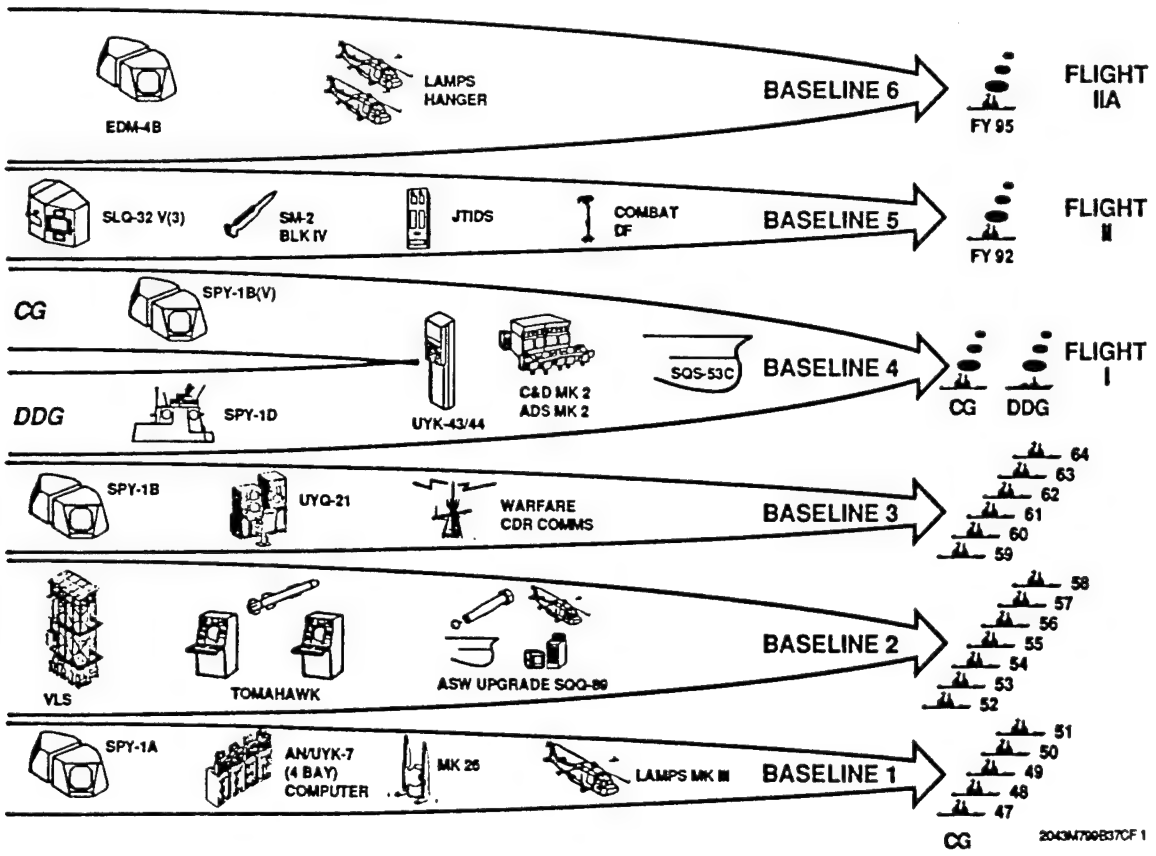



Fig. 2. AEGIS Combat System Evolution.

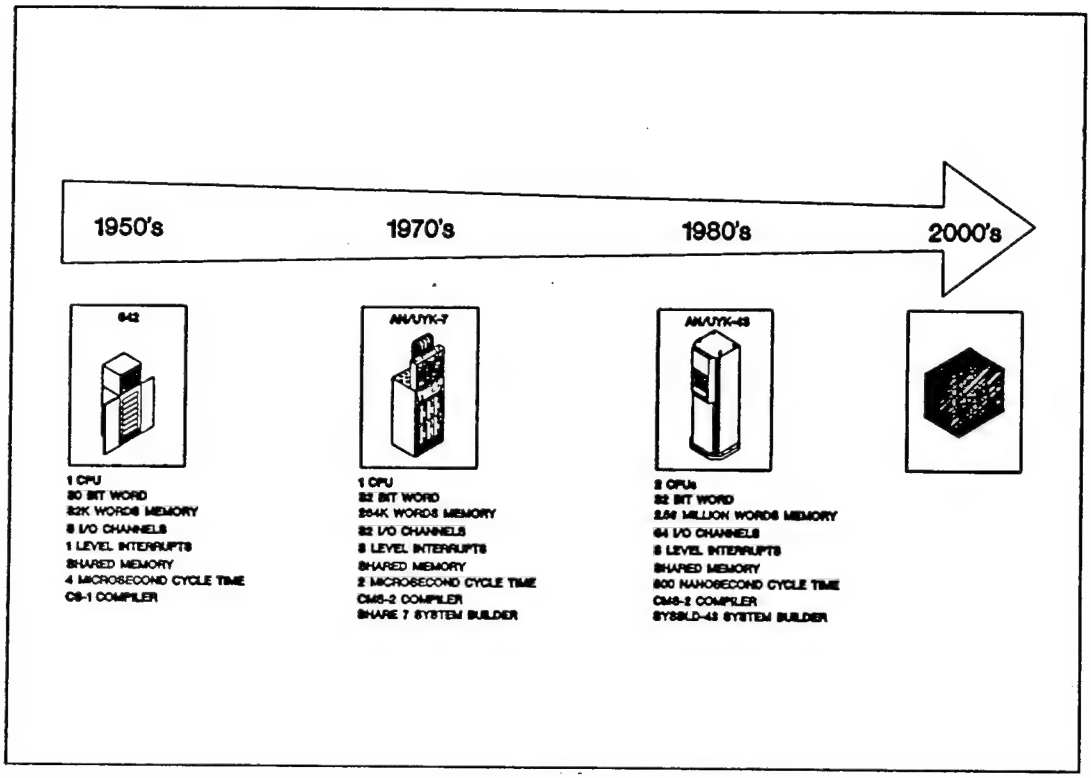



Fig. 3. Navy Standard Computer Evolution.

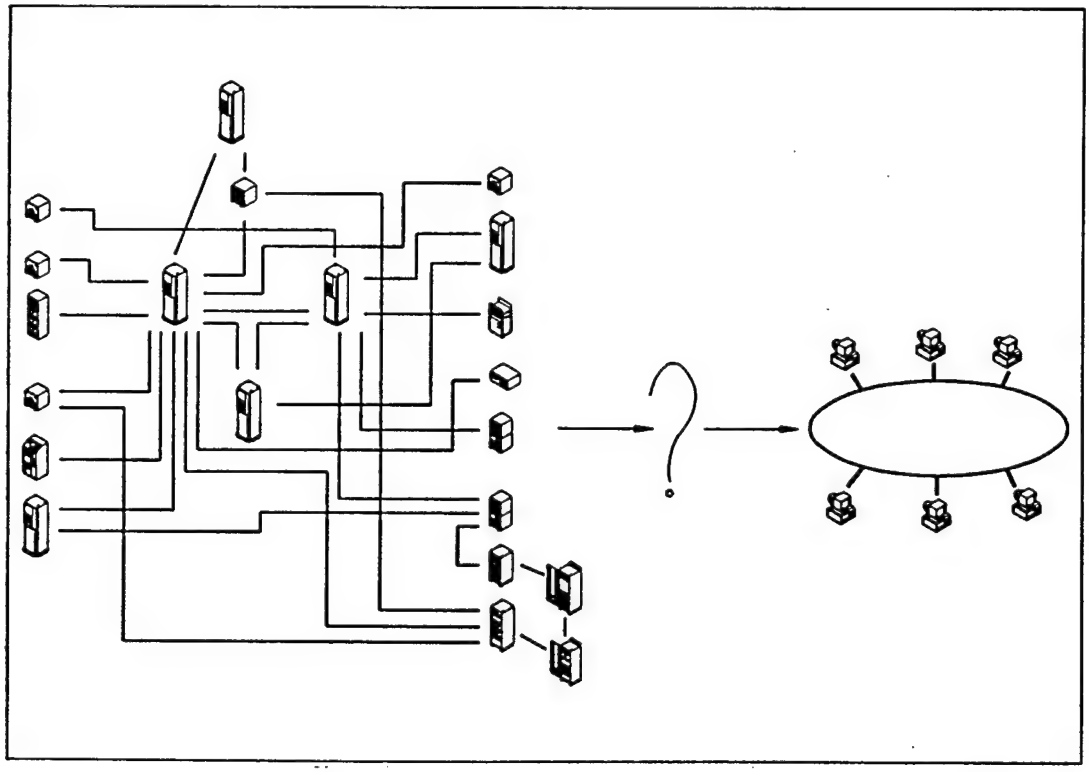


Fig. 4. Combat System Architecture Evolution.

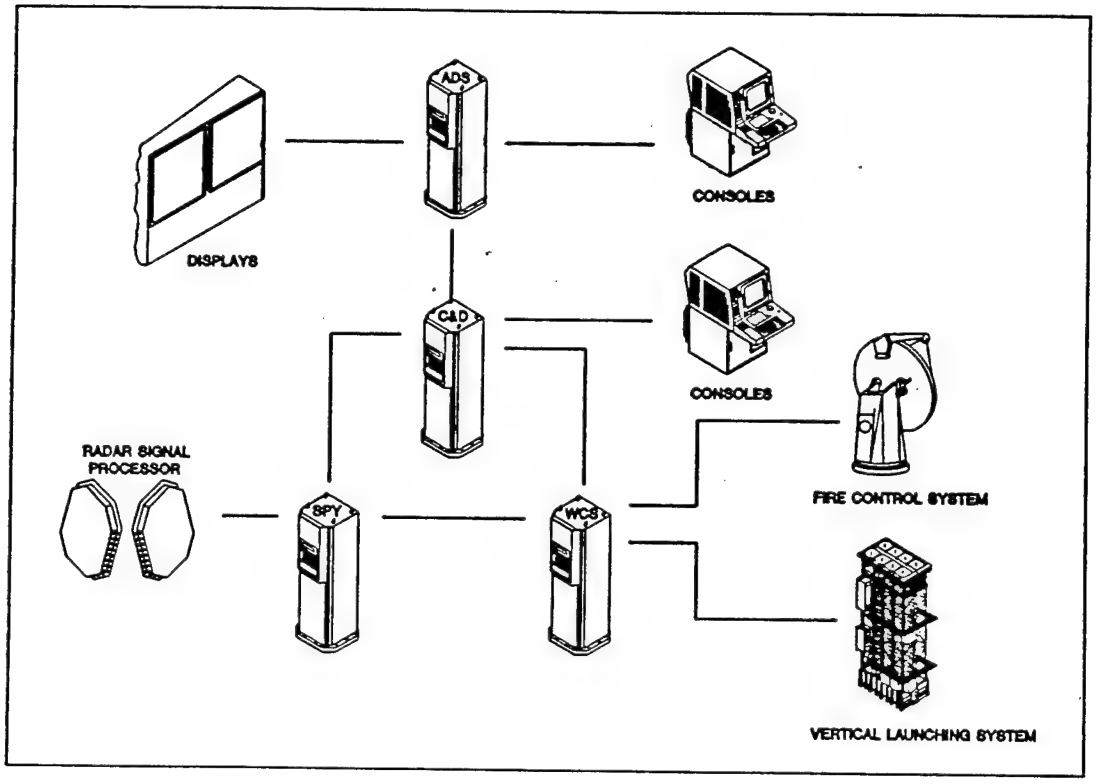



Fig. 5. Baseline 4 AN/UYK-43 Computer System.

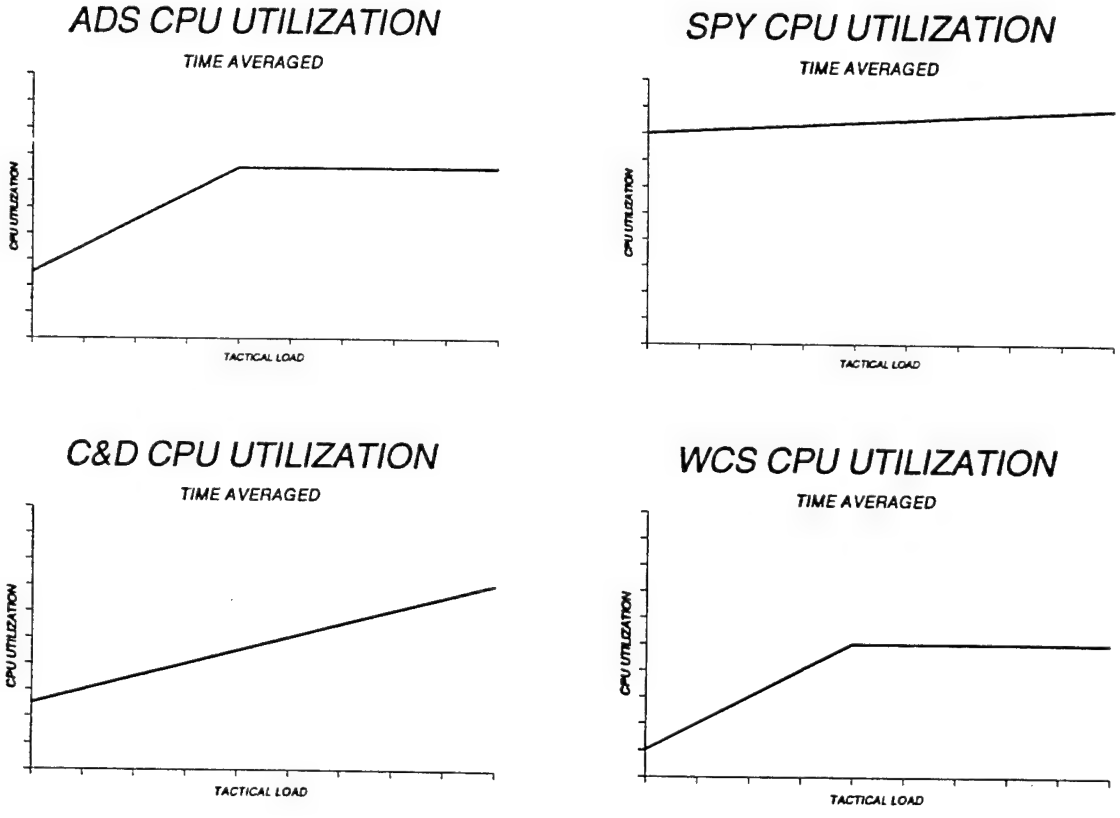



Fig. 6. Predicted Response to Tactical Load.

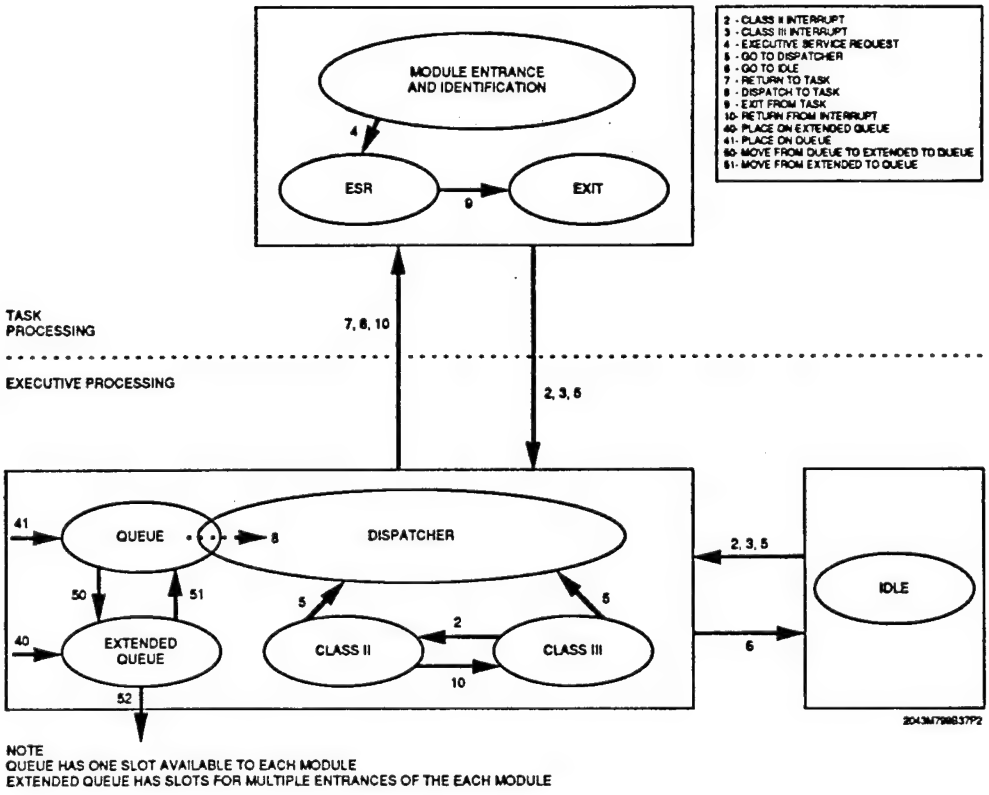



Fig. 7. Trace Event Model.

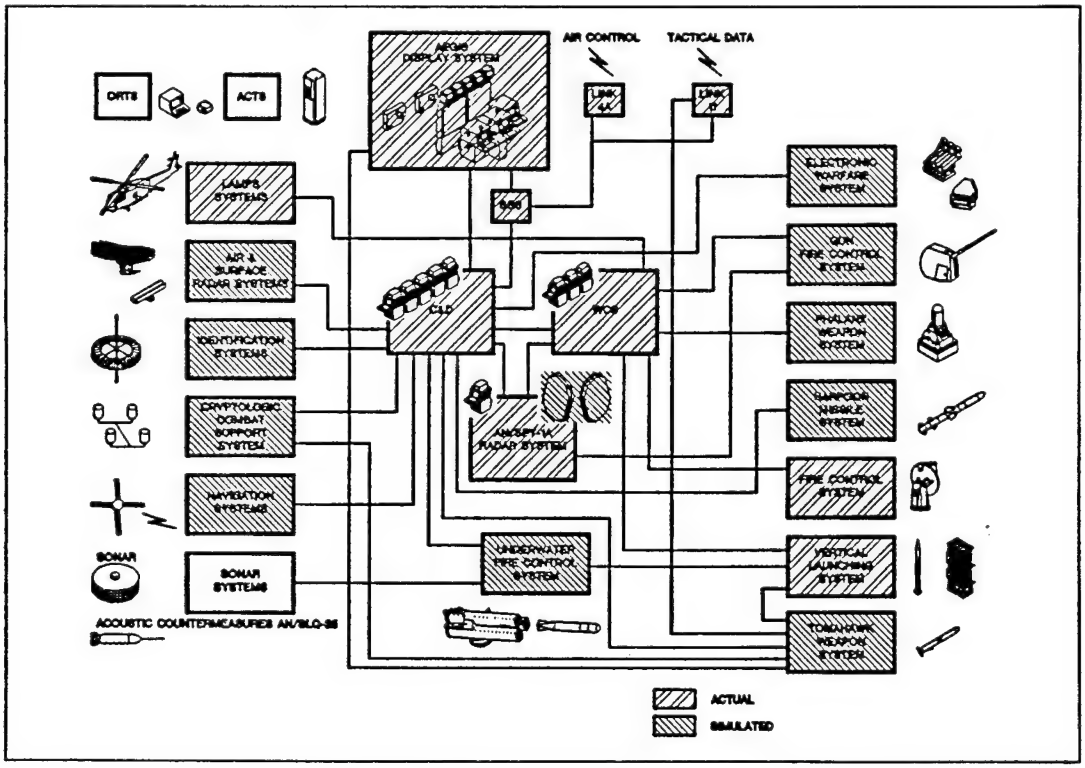



Fig. 8. Experimental Configuration.

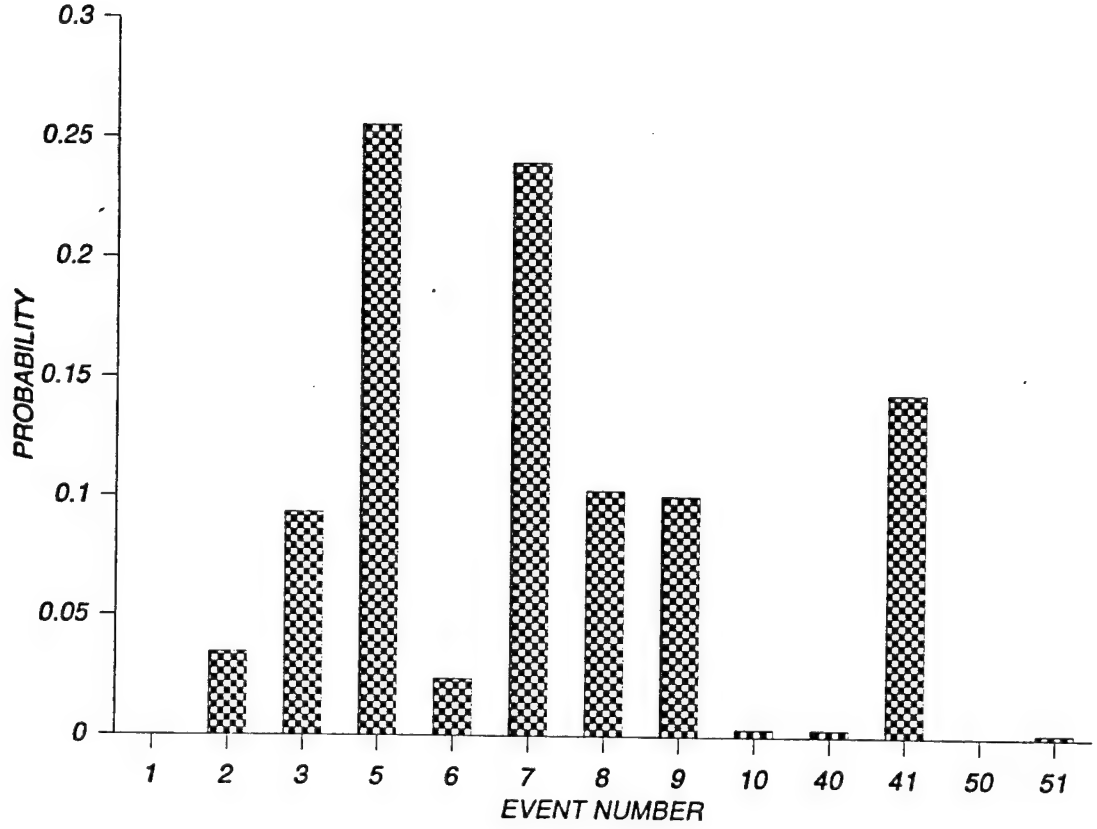



Fig. 9. Example of Events Distribution.

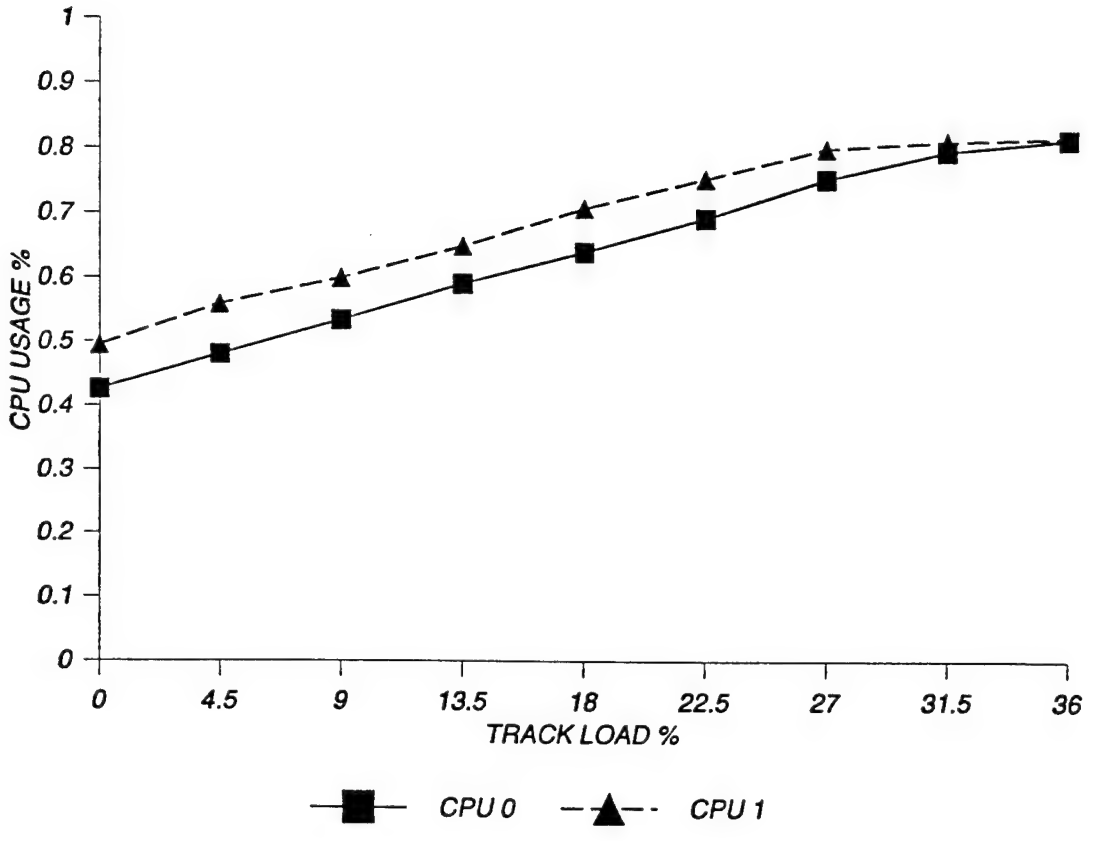



Fig. 10. CPU Response as a Function of Tactical Load.

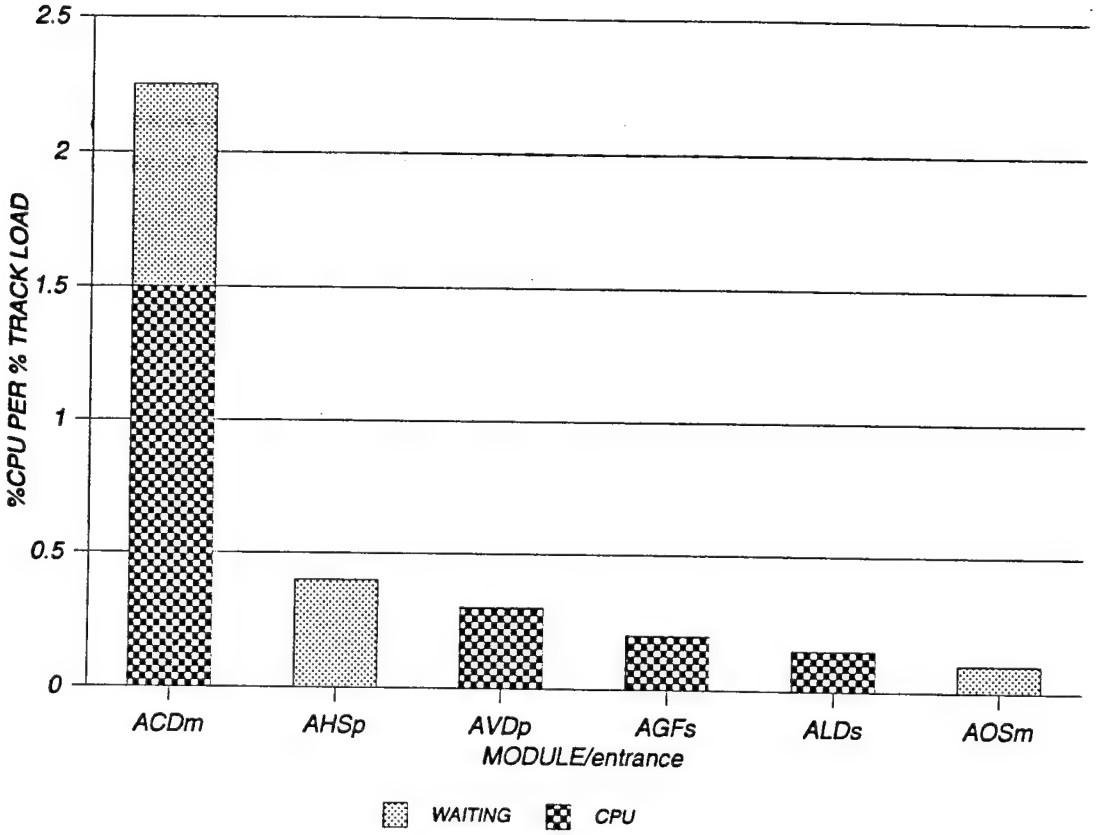



Fig. 11. Module Sensitivity to Track Load.

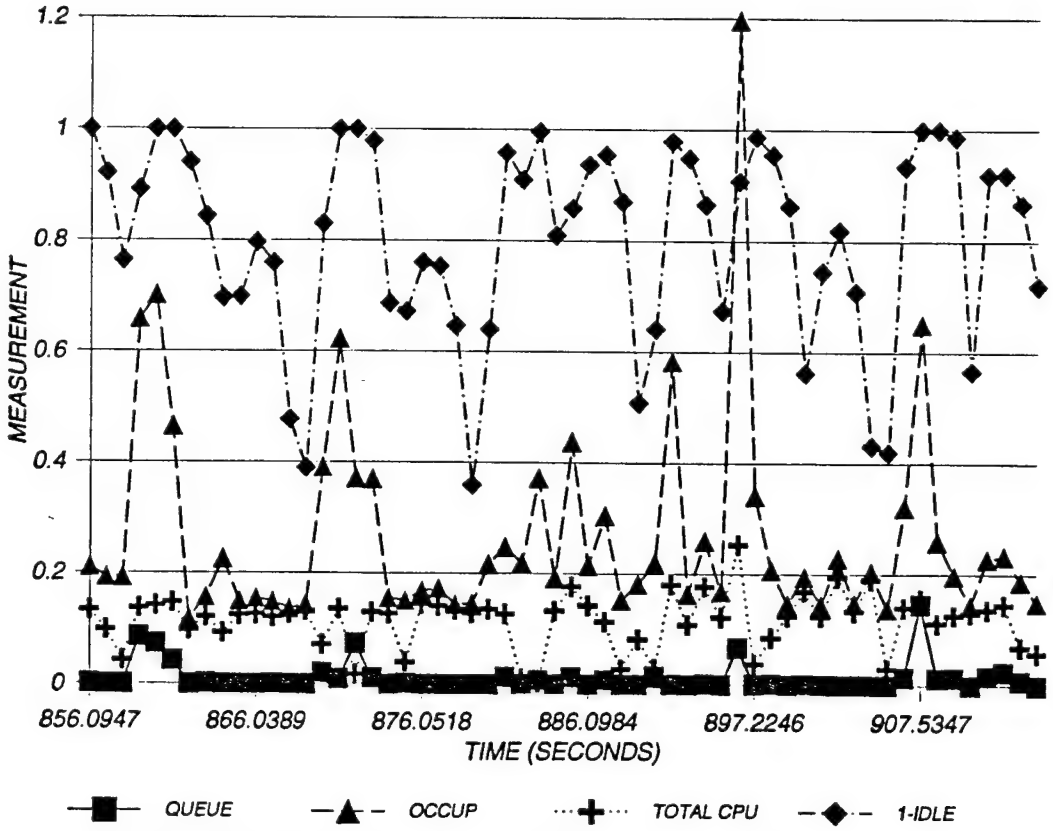



Fig. 12. Illustration of Correlations Between Measured Parameters.

# USING METRICS TO CONTROL AND PREDICT THE QUALITY OF SPACE SHUTTLE FLIGHT SOFTWARE

**Dr. Norman F. Schneidewind**

**Code SM/Ss**
**Naval Postgraduate School**
**Monterey, CA 93943**

**Voice: (408) 656-2719/2471**
**Fax : (408) 656-3407**

**Internet: schneidewind@nps.navy.mil**

## INTRODUCTION

Software quality metrics are effective in helping to assure the quality of software on large projects such as the Space Shuttle flight software. We show that it is possible to apply metrics to control and predict software quality based on metrics collected during design. The approach we use is to validate metrics against a quality factor (discrepancy reports) in accordance with the metrics validation methodology we developed [SCHa92] and that is included in the *IEEE Standard for a Software Quality Metrics Methodology (1061)* [IEE93]. Various statistical tests are conducted to see whether there is an association between the metrics and quality factor. In particular, we investigate the feasibility of controlling quality by using threshold metrics (i.e., critical values) and predicting quality by using metric predictor equations. In demonstrating the feasibility of these quality functions, we find it important to smooth the data in order to identify relationships between quality and metrics and to identify candidate metrics on the basis of the metrics set that has the highest dependence with quality and the highest independence from other sets of metrics. The results are significant for the Space Shuttle flight software in providing validated metrics for early identification of quality problems, and the methodology is applicable to other large scale software projects.

### Scope

Our emphasis is the validation of metrics for the Space Shuttle flight software. We provide examples of the validation process and the application of the validated metrics to the project data that were available for validation. The application of the validated metrics to *other* Space Shuttle software is the next step in our research and beyond the scope of this paper.

## OBJECTIVE

Our objective is to see whether it is possible to find relationships between metrics and quality factors so that we can make predictions of quality on large-scale projects such as the Space Shuttle. Our predictions take the following forms: 1) Boolean discriminator functions, based on metrics, that can predict during design whether the quality of the evolving product (e.g., module) is within the threshold of acceptable quality, and indicate whether remedial action is necessary (e.g., detailed inspection and tracking of the quality of the module during test and operation); and 2) regression equations, based on metrics, that can predict during design future *drcount* (counts of discrepancy reports written against modules of various software problems that may involve requirements, design, code, configuration, etc.)

## APPROACH

In order to find out whether we can predict software quality of the Space Shuttle flight software, we use the following approach:

- Apply quality factor-metric validity criteria that we developed previously to support the functions of quality control and prediction [IEE93, SCH92a].

- Use various statistical methods to answer the following questions: 1) Is there a relationship between one or more metrics

and the quality factor *drcount*? (i.e., are there independent vectors of metrics that are related to *drcount*?); 2) If there is a relationship, is there a clustering of *drcount* and metric values? If clustering exits, there is the potential for identifying metric discriminator functions and values to serve as thresholds for controlling quality during design.

- Smooth the data to identify the relationships. We find that data smoothing is essential to make sense of the data for 1489 modules. When we make plots of the raw data, no pattern emerges. However, when the data are divided into classes and the average and other statistics are computed for each class, relationships emerge.

- Apply categorical data analysis (non-parametric statistical techniques [CON71]), using the *Discriminative Power* validity criterion, to candidate metrics to analytically identify critical values to support the quality control function. A by-product of the search for quality discriminators is the discovery of regression equations, using the *Predictability* validity criterion, to support the quality prediction function.

## METRICS DISCRIMINATIVE POWER MODEL

### Discriminative Power Validation

Using our metrics validation methodology [SCH92a], and the Space Shuttle flight software metrics and discrepancy reports, we validate metrics with respect to the quality factor *drcount*. In brief, this involves conducting statistical tests to determine whether there is a high degree of association between a quality factor and metrics.

### Quality Control

The quality control function is used during design to flag software for detailed inspection that exceeds quality limits. Quality control is the evaluation of modules against predetermined critical values of metrics (e.g., has more than 8 *statements* or 9 *nodes*). The purpose of control is to allow software managers to identify software that exceeds the quality limits sufficiently early in the development process to take corrective action when the cost is low. The *Discriminative Power* validity criterion, which supports the quality control function, is defined as follows [SCH92a]:

### Discriminative Power

Given matrix $M_{ij}$ of **n** modules and **m** metrics (i.e., **nm** metric values), vector $M_{cj}$ of **m** metric critical values, vector $F_i$ of **n** quality factor values, and scaler $F_c$ of quality factor critical value, $M_{ij}$ must be able to discriminate with respect to $F_i$, for a specified $F_c$, as shown in the following relation:

$$\begin{aligned} M_{ij} > M_{cj} &\Longleftrightarrow F_i > F_c \text{ and} \\ M_{ij} \le M_{cj} &\Longleftrightarrow F_i \le F_c \end{aligned} \qquad (1)$$

for $i=1,2,\ldots,n$, and $j=1,2,\ldots,m$ with specified $\alpha$, where $\alpha$ is the significance level of a statistical test for estimating the degree to which (1) holds. In other words do the indicated metric relations imply corresponding quality factor relations in (1)?

This criterion assesses whether $M_{cj}$ has sufficient *Discriminative Power* to be capable of separating a set of high quality modules from a set of low quality modules. For example, would $M_{i1} > M_{c1}=8$ *statements* or $M_{i2} > M_{c2}=9$ *nodes* correspond to modules with *drcount*$>F_c=0$ (one or more discrepancy reports written against a module)? With this capability, we establish critical values for metrics that are used to identify modules that have unacceptable or questionable quality.

The desired quality level is set by the choice of $F_c$. This is the case because $M_{cj}$ is validated with respect to $F_c$. A low value of $F_c$ (high quality) would produce an $M_{cj}$ that would make it more difficult for modules to pass a quality check than a high value (low quality).

Having defined the *Discriminative Power* validity criterion, we now develop a model that will allow us to validate metrics for the purpose of controlling quality during software development. It is important to note that we use both quality factor data and metric data during validation but only metric data during application. The reason for this is that the validated

metrics are applied as indirect measures of quality until the time when direct measures of quality -- quality factors -- can be collected. During validation $F_i$ and $M_{ij}$ are collected from a sample of modules on a project that is related or similar to the application project. Thus evaluations of metrics obtained during validation are only estimates of how effective the metrics will be when they are applied. Once metrics have been validated, they are applied to modules on the application project that have not reached the phase when quality factor data are available. We use the validated metrics to make decisions about the quality of the product in lieu of having the quality factor data. Also it is important to recognize that validation is performed retrospectively. That is, with both $F_i$ and $M_{ij}$ in hand, we can evaluate how well $M_{ij}$ would have performed if it had been applied to these sets of data to make decisions about software quality.

Discriminative Power Validation Model

We use this model to validate $M_{cj}$ with respect to $F_c$, using *Contingency Table* analysis and the chi-square criterion [CON71], and other criteria: module misclassification, inspection required, and product quality [SCHb92]. Equation (2) gives the module count, based on Boolean functions of $F_i$ and $M_{ij}$, that are calculated over the **n** modules for **m** metrics. This equation is an implementation, with specific quality factor and metrics, of the relation given in (1).

$$C_{11}=\underset{i=1}{\overset{n}{\text{COUNT}}}\ \text{FOR}\ ((F_i\le F_c)\wedge(M_{i1}\le M_{c1})\ldots\wedge(M_{ij}\le M_{cj})\ldots\wedge(M_{im}\le M_{cm}))$$

$$C_{12}=\underset{i=1}{\overset{n}{\text{COUNT}}}\ \text{FOR}\ ((F_i\le F_c)\wedge((M_{i1}>M_{c1})\ldots\vee(M_{ij}>M_{cj})\ldots\vee(M_{im}>M_{cm})))$$

$$C_{21}=\underset{i=1}{\overset{n}{\text{COUNT}}}\ \text{FOR}\ ((F_i>F_c)\wedge(M_{i1}\le M_{c1})\ldots\wedge(M_{ij}\le M_{cj})\ldots\wedge(M_{im}\le M_{cm}))$$

$$C_{22}=\underset{i=1}{\overset{n}{\text{COUNT}}}\ \text{FOR}\ ((F_i>F_c)\wedge((M_{i1}>M_{c1})\ldots\vee(M_{ij}>M_{cj})\ldots\vee(M_{im}>M_{cm}))) \qquad (2)$$

for j=1,...,m, and where COUNT(i)=COUNT(i-1)+1 FOR Boolean expression *true* and COUNT(i)=COUNT(i-1), otherwise; COUNT(0)=0.

The purpose of these counts is to identify $M_{cj}$ for a given $F_c$, where $F_c$ is inversely related to quality level (e.g., if we want maximum quality, we set $F_c=0$). The counts correspond to the cells of the *Contingency Table*, as shown in Table 1, where row and column totals are also shown. A special case of equation (2) is the use of a single metric. The *OR* function is used so that a module would not be acceptable unless it satisfies all **m** critical values. It is important to understand that during validation we know, by having inspected or executed the modules, which have $F_i>F_c$ (e.g., *drcount*>0) and which do not. We also collect the metrics data. Therefore, for each module we are able to classify it into one of the four categories shown in Table 1.

Table 1

Validation Contingency Table

| | $\wedge(M_{ij}\le M_{cj})$ | $\vee(M_{ij}>M_{cj})$ | |
|---|---|---|---|
| $F_i\le F_c$ High Quality | $C_{11}$ | Type 2 $C_{12}$ | $n_1$ |
| $F_i>F_c$ Low Quality | Type 1 $C_{21}$ | $C_{22}$ | $n_2$ |
| | $N_1$ | $N_2$ | $n$ |

We validate vector $M_{cj}$ by demonstrating that it partitions Table 1 in such a way that $C_{11}$ and $C_{22}$ are large relative to $C_{12}$ and $C_{21}$. If this is the case, a large number of high quality modules (e.g., modules with zero *drcount*) would have

$M_{ij} \leq M_{cj}$ and would be correctly classified as high quality. Similarly, a large number of low quality modules (e.g., modules with non-zero *drcount*) would have $M_{ij} > M_{cj}$ and would be correctly classified as low quality. The degree to which this is the case is estimated statistically by the chi-square ($\chi^2$) statistic [CON71]. If calculated $\chi^2_c > \chi^2_s$ (chi-square at specified $\alpha_s$) and if calculated $\alpha_c < \alpha_s$, we conclude that $M_{cj}$ is statistically significant. In addition it is important to estimate *Discriminative Power* from an application perspective. This evaluation follows under the categories of Misclassification, Inspection, and Quality.

### Misclassification

The *Discriminative Power* of $M_{cj}$ is estimated by noting in **Table 1** that ideally $C_{11}=n_1=N_1$, $C_{12}=0$, $C_{21}=0$, $C_{22}=n_2=N_2$. The extent that this is not the case is estimated by *Type 1* misclassifications (i.e., the module has *Low Quality* and the metrics "say" it has *High Quality*) and *Type 2* misclassifications (i.e., the module has *High Quality* and the metrics "say" it has *Low Quality*). Thus we define the following measures of misclassification:

Proportion of Type 1: $P_1 = C_{21}/N$ **(3)**

Proportion of Type 2: $P_2 = C_{12}/N$ **(4)**

Proportion of Type 1+Type 2: $P_{12} = (C_{21} + C_{12})/N$ **(5)**

Note that *High Quality* and *Low Quality* are relative terms. The fact that a module is classified as *Low*, does not necessarily mean that it is poor quality in absolute terms; it only means it is *Low* relative to *High*. Furthermore, because validated metrics are not perfect classifiers of quality, a module classified as *Low*, when the metrics are applied, does not mean necessarily that the module will have low quality during operation; conversely, a classification of *High* is no guarantee of high quality. The benefit of metrics is the early indication of *potential* quality problems so that corrective action can be taken, if necessary, when the cost of correction is relatively low.

### Inspection

Inspection is one of the costs of quality. We are interested in weighing inspection requirements against the quality that is achieved for various values of $M_{cj}$. This is another way to estimate *Discriminative Power*. We estimate inspection requirements by noting that all modules with $M_{ij} > M_{cj}$ must be inspected; this is the count $C_{12} + C_{22}$. Thus the proportion of modules that must be inspected is given by:

$$I = (C_{12} + C_{22})/N \quad \textbf{(6)}$$

Part of this inspection is "wasted" because of *Type 2* ($C_{12}$) misclassifications (i.e., modules are inspected because they are incorrectly flagged). We estimate wasted inspection by using equation (4) for *Type 2* misclassifications. *Discriminative Power* is also estimated by the ratio of useful to wasted inspection given by:

$$RI = C_{22}/C_{12} \quad \textbf{(7)}$$

### Quality

Lastly, *Discriminative Power* is estimated by summing remaining quality factor RF (e.g., remaining *drcount*), given by equation (8). This is the sum of $F_i$ not caught by inspection because $\wedge(M_{ij} \leq M_{cj})$ for these modules.

$$RF = \sum_{i=1}^{n} F_i \text{ FOR } ((M_{i1} \leq M_{c1}) \ldots \wedge (M_{ij} \leq M_{cj}) \ldots \wedge (M_{im} \leq M_{cm})) \quad \textbf{(8)}$$

for $j = 1, \ldots, m$.

The proportion remaining is estimated by equation (9), where TF is the total $F_i$ prior to inspection.

$$RFP = RF/TF \quad (9)$$

The density remaining is estimated by equation (10).

$$RFD = RF/n \quad (10)$$

In addition we estimate the count of modules remaining that have $F_i > 0$. This is calculated from $C_{21}$ with $F_i > F_c = 0$ (see equation (2) and Table 1); the proportion remaining RMP is given by equation (11). Note that RMP=$P_I$ (proportion of *Type 1* misclassifications) when $C_{11}$ is calculated with $F_i = F_c = 0$.

$$RMP = C_{21}/n, \text{ for } F_i > F_c = 0 \quad (11)$$

In order to start the validation process, we select a subset from the set of **m** metrics. We do this by identifying the subset that has the greatest association with *drcount*, as determined from statistical analysis, as described in the next section. Then values of $M_{ij}$ are selected to begin the estimation of *Discriminative Power* as described above. We select these values from the data by pairing $M_{ij}$ with the chosen $F_c$, as we will explain.

## Statistical Analysis

### Data

The thirteen metrics that were collected with the quality factor *drcount*, for 1489 modules written in HAL/S, are defined as follows:

Metrics:

| | |
|---|---|
| eta1: | unique operator count |
| eta2: | unique operand count |
| n1: | total operator count |
| n2: | total operand count |
| stmts: | total statement count |
| loc: | total non-commented lines of code |
| comments: | total comment count |
| nodes: | total node count (in control graph) |
| edges: | total edge count (in control graph) |
| paths: | total path count (in control graph) |
| cycles: | total cycle count (in control graph) |
| maxpath: | maximum path length (edges in control graph) |
| avepath: | average path length (edges in control graph) |

### Procedure

The procedure we describe is used to select the candidate metrics (i.e., metrics that will be tested against validity criteria for their ability to act as discriminators and predictors of quality). Scatter diagrams and histograms are very useful to obtain a feel for the data (i.e. What are the distributions, both univariate and multivariate? Are the distributions normal or skewed? Is there a pattern? Is there a bunching of the data in certain ranges of the data?). *Principal Components* and *Factor Analysis* assist in identifying independent vectors of metrics that are related to *drcount*.

### Principal Components

In applying *Principal Components* to metrics, our objective is to reduce a large number of metrics to a few weighted linear combinations of metrics that are: 1) independent and 2) account for maximum variation in the metrics [JOB92, MUN93]. Given metric matrix $M_{ij}$ (**n** modules, **m** metrics), a matrix of weights $W_{ij}$ (**m** metrics, **m** components) is determined such that the *Principal Components* matrix $P_{ij}$ (**n** modules, **m** components) in equation (12) satisfies 1) and 2),

where number of metrics equals number of components.

$$P_{ij}=M_{ij}W_{jj} \quad (12)$$

We look for metrics that have the highest weights on a given component, preferably *Component 1*, which accounts for the largest variation in the metrics. In addition, we want metrics that have the highest correlation with the quality factor *drcount*. Two very useful plots that serve these purposes are shown in Figures 1 and 2, respectively. Figure 1 shows that *stmts* and *nodes* have the largest positive weights on *Component 1*, .319 and .311, respectively and low or negative weights on *Component 2*, .108 and -.228, respectively. This is desirable because *Components 1* and *2* are supposed to be orthogonal.

In Figure 2, the lengths of the lines from the origin represent the contribution of the metrics to the principal components (i.e., the horizontal component represents the contribution to *Component 1* and the vertical component represents the contribution to *Component 2*). We observe the large contributions of *drcount*, *stmts*, and *nodes* to *Component 1* relative to small contributions to *Component 2* (negative for *nodes*). Also the angle between any pair of lines is inversely proportional to the correlation between them (i.e., if two metrics were on the same line, they would have 100% correlation). Note the small angle between *drcount* and *stmts*.

### Factor Analysis

We also use *Factor Analysis*, a statistical analysis procedure related to *Principal Components*, where our purpose is to group metrics into subsets that are highly correlated and have low correlations with other subsets (these subsets are called *factors*) [JOB92]. This analysis confirms that *stmts* and *nodes* are the best candidate metrics for further analysis (i.e., have the highest correlations (*factor loadings*) with *Factor 1* of .870 and .849, respectively).

Although other metrics -- *edges*, *maxpath*, and *avepath* -- also score high on both *Principal Components* and *Factor Analysis*, *stmts* and *nodes* are used in the validation tests, with the option of including more metrics in the analysis at a later date.

### Metric Clusters

If *stmts* and *nodes* are to serve as discriminators of quality we want to know whether there is clustering of values. If clustering occurs, it tells us (approximately) where to set $S_c$ and $N_c$, which are the critical values of *stmts* and *nodes*, respectively. We see from Figures 3, 4, and 5 that *drcount*, *stmts*, and *nodes* cluster at low values. Thus if $D_c$, the critical value of *drcount*, is set equal to 0, $S_c$ and $N_c$ must be set to low values to force a high percentage of the modules with $D_c > 0$ to fail the quality check.

### Data Smoothing

A critical part of the metrics analysis procedure is to smooth the data in order to obtain meaning from them. To illustrate the importance of this we first plot *drcount* against *stmts* for 1489 modules in Figure 6. We cannot make sense of this plot. We do note that there are modules with zero *stmts*. These are assembly language modules with no HAL/S statements that apply to them; there are 92 of these modules. Only the remaining 1397 non-zero *stmts* modules with total *drcount* (TF) of 2579 are considered in subsequent analyses. Trends in the data are seen when we divide the data into classes and compute the *average* (the *range* and *standard deviation* are also given), as shown in Table 2. The modules in each class are the same for *stmts*, *nodes*, and *drcount*. We see the trends when plots are made of *avestmts*, *avenodes*, and *avedrcount*, as shown in Figures 7, 8, and 9. The twelve classes account for 97.7 % of the modules. In addition to helping us find critical values of metrics for use in quality control, data smoothing allows us to produce predictor equations as will be seen shortly.

**Table 2**

**Smoothed Metrics Data**

| Class | *stmts* | | | *nodes* | | | *dr count* | | |
|---|---|---|---|---|---|---|---|---|---|
| | Range | Ave | S.D. | Range | Ave | S.D. | Range | Ave | S.D. |
| 1 | 1-34 | 8.97 | 9.18 | 3-288 | 9.86 | 20.89 | 0-31 | .65 | 2.17 |
| 2 | 35-68 | 48.72 | 10.14 | 3-60 | 21.82 | 13.68 | 0-13 | 1.57 | 2.33 |
| 3 | 69-103 | 85.12 | 10.62 | 3-79 | 35.35 | 18.91 | 0-13 | 2.08 | 2.57 |
| 4 | 104-137 | 119.58 | 9.30 | 5-119 | 45.95 | 28.10 | 0-18 | 2.79 | 3.84 |
| 5 | 138-171 | 156.55 | 9.28 | 5-147 | 56.13 | 39.34 | 0-22 | 4.00 | 4.43 |
| 6 | 172-206 | 189.70 | 10.83 | 5-167 | 75.08 | 44.02 | 0-13 | 3.95 | 4.15 |
| 7 | 207-240 | 222.77 | 9.48 | 5-156 | 90.64 | 40.14 | 0-13 | 4.91 | 3.22 |
| 8 | 241-275 | 254.37 | 11.50 | 5-166 | 71.04 | 61.04 | 0-12 | 3.85 | 4.09 |
| 9 | 276-309 | 294.20 | 10.56 | 5-187 | 95.67 | 56.17 | 0-34 | 5.67 | 8.25 |
| 10 | 310-343 | 320.22 | 10.47 | 5-171 | 65.33 | 59.04 | 0-10 | 4.22 | 3.90 |
| 11 | 344-378 | 357.86 | 9.70 | 5-338 | 156.00 | 81.61 | 1-37 | 9.93 | 9.36 |
| 12 | 379-412 | 397.88 | 6.22 | 5-232 | 145.25 | 94.17 | 1-26 | 10.38 | 9.55 |

Discriminative Power Validation Model Application

Now we apply the *Discriminative Power Validation Model* to develop discriminators for controlling the quality of the Space Shuttle flight software. Using equations (2) and (8) and the following notation for the quality factor and candidate metrics, we develop equations (13) and (14):

$D_i$: observed *drcount* for module i

$D_c$: calculated critical value of *drcount*

$S_i$: observed *stmts* for module i

$S_c$: calculated critical value of *stmts*

$N_i$: observed *nodes* for module i

$N_c$: calculated critical value of *nodes*

$$C_{11}=\underset{i=1}{\overset{n}{\text{COUNT}}}\ \text{FOR}\ ((D_i \le D_c) \wedge (S_i \le S_c) \wedge (N_i \le N_c))$$

$$C_{12}=\underset{i=1}{\overset{n}{\text{COUNT}}}\ \text{FOR}\ ((D_i \le D_c) \wedge ((S_i > S_c) \vee (N_i > N_c)))$$

$$C_{21}=\underset{i=1}{\overset{n}{\text{COUNT}}}\ \text{FOR}\ ((D_i > D_c) \wedge (S_i \le S_c) \wedge (N_i \le N_c))$$

$$C_{22}=\underset{i=1}{\overset{n}{\text{COUNT}}}\ \text{FOR}\ ((D_i > D_c) \wedge ((S_i > S_c) \vee (N_i > N_c))) \qquad \textbf{(13)}$$

$$RF=\sum_{i=1}^{n} D_i\ \text{FOR}\ ((S_i \le S_c) \wedge (N_i \le N_c)) \qquad \textbf{(14)}$$

We apply equations (3-11) and *Contingency Table* analysis to calculate the entries in **Table 3.** Then we use **Table 3** to evaluate the *Discriminative Power* of $S_i > S_c$, $N_i > N_c$, and $(S_i > S_c) \vee (N_i > N_c)$ with respect to the quality factor *drcount* ($D_i$). The values 8 and 9 are the averages of $S_c$ and $N_c$, respectively, in **Table 2** that correspond to $D_c \cong 0$; the values 48 and 21 correspond to $D_c \cong 1$; and the values 85 and 35 correspond to $D_c \cong 2$. We note that *drcount* is used only during validation to validate $S_c$ and $N_c$; *drcount* data are not available when the validated $S_c$ and $N_c$ are applied. When $S_c$ or $N_c$ appear alone in **Table 3,** it means it is used as a single discriminator. When both appear, it means they are members of an *OR* discriminator function. Because $\chi^2_c \gg \chi^2_s$ and $\alpha_c \ll \alpha_s$ ($\chi^2_s = 10.83$, $\alpha_s = .001$, $\alpha_c = 0$ to five places) in all cases in **Table 3,** all functions are easily validated statistically. We note that an *OR* function has higher chi-square than individual $S_c$ or $N_c$, for the same critical value pair, and that the *OR* function with the smallest $S_c$,$N_c$ pair has the highest chi-square of all. Although these results validate the metrics statistically, we must also evaluate *Discriminative Power* from an application perspective.

We apply **Table 3** by observing that if we use small values of $S_c$ and $N_c$, we should expect to achieve high quality with high inspection requirements; conversely, if we use large values, we should expect low quality and low inspection requirements. Also we see that the *OR* function achieves higher quality with higher inspection requirements than single metrics for a given $S_c$ and $N_c$ pair. We see these results in Figure 10 where quality (RFP, RMP) and inspection (I) are plotted against $S_c$,$N_c$ of $(S_i > S_c) \vee (N_i > N_c)$. An alternate way of showing these relationships is Figure 11 where quality is plotted against inspection for various $S_c$,$N_c$. Note that selecting small values of $S_c$ and $N_c$ does not necessarily mean that all modules should be designed small; there may be good reasons for exceeding the critical values. Rather, selecting small critical values provides a fine net that catches modules that are likely to cause quality problems later in the software process.

We decide on the values of $S_c$ and $N_c$ to use in an application by weighing quality against inspection. For example, if we want maximum quality (e.g., RFP=4.11%) for the Space Shuttle flight software, with little consideration of cost of inspection (I=67.9%), we would select $(S_i > 8) \vee (N_i > 9)$. On the other hand, if inspection must be less than 50%, and a quality of RFP=18.1 % is acceptable, we would select $(S_i > 48) \vee (N_i > 21)$. Whichever $S_c$,$N_c$ pair is selected, it would be applied to *other similar* software with the expectation that in a high percentage of the modules the pair would be able to flag ones that constitute a quality risk. We note that the selected $S_c$,$N_c$ pair would not necessarily be used indefinitely. As experience with the validated metrics is obtained on projects and as new quality factor and metrics data are collected, the metrics would be revalidated.

**Table 3**

*Discriminative Power* **Validity Evaluation**

| $D_c$ | $S_c$ | $N_c$ | $P_1$ | $P_2$ | $P_{12}$ | I | RI | RFP | RFD | RMP | $\chi^2_c$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 8 | - | 4.72 | 27.3 | 32.0 | 63.8 | 1.34 | 9.89 | .183 | 4.72 | 258 |
| 0 | - | 9 | 12.6 | 17.8 | 30.4 | 46.4 | 1.61 | 28.6 | .528 | 12.6 | 208 |
| 0 | 8 | 9 | 2.79 | 29.5 | 32.3 | 67.9 | 1.31 | 4.11 | .0759 | 2.79 | 286 |
| 1 | 48 | 21 | 7.23 | 20.0 | 27.2 | 42.3 | 1.11 | 18.1 | .335 | 12.0 | 261 |
| 2 | 85 | 35 | 6.73 | 16.8 | 23.5 | 31.3 | .86 | 27.8 | .513 | 18.2 | 237 |

$D_c$: Calculated critical value of *drcount*
$S_c$: Calculated critical value of *stmts*
$N_c$: Calculated critical value of *nodes*
$P_1$: Percentage of Type 1 misclassifications
$P_2$: Percentage of Type 2 misclassifications
$P_{12}$: Percentage of Type 1+Type 2 misclassifications
I: Percentage of modules inspected
RI: Ratio of useful to wasted inspection
RFP:Percentage of *drcount* remaining after inspection
RFD:Density of *drcount* remaining after inspection (*drcount*/module)
RMP:Percentage of modules remaining after inspection with *drcount*>0
$\chi^2_c$: Calculated chi-square

## METRICS PREDICTION MODEL

### Predictability Validation

#### Quality Prediction

Quality prediction is a forecast of the quality factor vector $F_i$ as a function of the metric matrix $M_{ij}$. Validated metrics are used during the design phase to make predictions of test or operational phase quality factors (e.g., *drcount*). The purpose of prediction is to provide software managers with a forecast of the quality of the operational software and to flag modules for detailed inspection whose predicted quality factor values exceed target values. Quality prediction differs from quality control in that the former predicts $F_i$ from $M_{ij}$, whereas the latter uses $M_{ij}$ to classify modules into quality categories (there is no prediction of quality). The *Predictability* criterion, which supports the quality prediction function, is defined as follows [SCH92]:

#### Predictability

A function $f(M_{ij})$ must be able to predict $F_i$ with the following accuracy:

$$\frac{\sum_{i=1}^{n} \frac{|F_i - f(M_{ij})|}{F_i}}{n} < \beta \qquad (14)$$

for $j=1,\ldots,m$, where n is number of modules, m is the number of metrics, and $\beta$ is the accuracy target. In effect, this is a mean relative error criterion.

### Predictability Validation Model

We use this model to validate $M_{ij}$ with respect to $F_i$, using regression analysis and the criteria of mean relative error (MRE), mean square error (MSE), mean residual (MR), and residual plots [KHO92, SCH93]. Although the *Predictability* criterion is based on MRE, as shown above, we use additional criteria, such as MSE, to evaluate alternate models of $f(M_{ij})$.

### Statistical Analysis

The candidate metrics *stmts* and *nodes* were identified in previous statistical analysis (e.g., *Principal Components*). Also it was shown that data smoothing (i.e., computing averages in classes of data) allowed trends to emerge in the data that were not apparent in analyzing the raw data. These results have implications for identifying $f(M_{ij})$. A look at Figures 7, 8, and 9, suggests the following about *avedrcount*: non-linear in *avestmts*, linear in *avenodes*, and non-linear in *avestmts* and *avenodes* used together.

#### Regression

Based on these *apparent* relationships, the following linear and non-linear regression models, equations (15), (16), and (17) are developed and evaluated, using the averages of the twelve classes of data in **Table 2**:

$$D_a(s) = \exp(.242 + .00523S_a) \qquad (15)$$

$$D_a(n) = -.262 + .0658N_a \qquad (16)$$

$$D_a(sn) = \exp(.348 + .00194S_a + .00826N_a) \qquad (17)$$

where the following notation is used in the regression equations and in the evaluations of the equations as predictors of quality:

$S_a$: *avestmts* used to produce $D_a(s)$ and $D_a(sn)$ or given value in $D_a(s)$ and $D_a(sn)$ used as predictors

$N_a$: *avenodes* used to produce $D_a(n)$ and $D_a(sn)$ or given value in $D_a(n)$ and $D_a(sn)$ used as predictors

$d_a$: *avedrcount* used to produce $D_a(s)$, $D_a(n)$, and $D_a(sn)$

$D_a(s)$: predicted *avedrcount* as a function of *avestmts*

$D_a(n)$: predicted *avedrcount* as a function of *avenodes*

$D_a(sn)$: predicted *avedrcount* as a function of *avestmts* and *avenodes*

$D_a'$: actual *avedrcount*

Equations (15), (16), and (17) are plotted and compared with *avedrcount* in Figures 12, 13, and 14, respectively.

## Predictability Validation Model Application

Now we apply the *Predictability Validation Model* to develop predictors of quality of the Space Shuttle flight software. In the linear equation (16), we find that including $S_a$ makes no improvement in *Predictability*; therefore the single metric $N_a$ is used. This is reflected in the following partial correlation coefficient matrix (coefficients that measure the *linear* association of a dependent variable and one of the independent variables but controls for the effects of the other independent variables [KLE78]):

*avedrcount* vs. *avestmts*: .030
*avedrcount* vs. *avenodes*: .909
*avestmts* vs. *avenodes*: .351

However the non-linear equation (17), which includes $S_a$, is competitive with the linear equation (16), as we see in Table 4.

**Table 4**

***Predictability* Validity Criterion**

| | MRE | MRE SD | MSE | MR | MR SD |
|---|---|---|---|---|---|
| $D_a(s)$ | .247 | .301 | 1.104 | .0151 | 1.097 |
| $D_a(n)$ | .127 | .117 | .281 | -.0000768 | .554 |
| $D_a(sn)$ | .192 | .388 | .198 | -.0300 | .463 |

MRE: Mean Relative Error
MRE SD: MRE Standard Deviation
MSE: Mean Square Error
MR : Mean Residual
MR SD: MR Standard Deviation

Three evaluators of goodness of fit of $D_a(s)$, $D_a(n)$, and $D_a(sn)$ are shown in Table 4: MRE, MSE, and MR. These are defined in equations (18), (19), and (20), respectively. In addition, the standard deviations are shown for MRE and MR. For MRE -- the *Predictability* criterion -- equation (18) is a modification of expression (14) to reflect the use of $D_a$ predictors and class averages.

$$MRE = \frac{\sum_{k=1}^{C} \frac{|d_{ak} - D_{ak}|}{d_{ak}}}{C} \quad (18)$$

$$MSE = \frac{\sum_{k=1}^{C} (D_{ak} - d_{ak})^2}{C} \quad (19)$$

$$MR = \frac{\sum_{k=1}^{C} (d_{ak} - D_{ak})}{C} \quad (20)$$

where **C** is the number of classes of metric and quality factor data for which averages are computed (e.g., twelve classes, as in **Table 2**).

*Mean Relative Error* is useful because it measures prediction error relative to observed *avedrcount*. *Mean Square Error* minimizes the sum of the variance of predicted *avedrcount* and the square of the bias (predicted-actual) *avedrcount*: $Var[D_a] + (D_a - D_a')^2$ [JEN68], where $D_a'$ is actual *avedrcount*. *Mean Residual* measures the mean of (observed-predicted) *avedrcount*, without regard to sign. Residual plots are important because they show whether there is stability in the predictions (i.e., residuals should be distributed uniformly around the zero residual line with approximately constant value).

The results in **Table 4** show there is no one predictor that excels in all criteria. The $D_a(s)$ predictor has a large MRE: 24.7%. The $D_a(n)$ predictor is better than $D_a(sn)$ with respect to MRE and MR, but worse with respect to MSE. Lastly, when the residuals are plotted, it is observed that the $D_a(sn)$ plot was the most stable.

We will retain all three predictors for predicting Space Shuttle flight software quality for the following reasons:

- There is no clear winner.
- It is desirable to have several predictors. Rather than relying on a single prediction, we make several predictions with the objective of bounding the actual *avedrcount*.
- In the early stages of development only *stmts* may be available, requiring the use of $D_a(s)$.
- We want to obtain experience with the predictors on the Space Shuttle flight software to evaluate how accurately the predictors perform *with data other than the data that was used to produce the predictors*, and we want to refine and revalidate the predictors as experience dictates.

The importance of the last point is highlighted in **Table 5** which shows the results of an experiment where the lower and upper limits of *stmts* and *nodes* are selected randomly from the 1397 modules, $S_a$ and $N_a$ are computed within the ranges, and predictions of *avedrcount* are compared with the actual values ($D_a'$). We see that $D_a(s)$ was the winner in two of the three cases and $D_a(n)$ was the loser in all three cases. Obviously more than three cases is necessary to produce a definitive result but the experiment illustrates the point of how results in the *prediction domain* can differ from results in the *model formulation domain*. Also we see that, as stated above, the three model predictions bound the actual value.

Table 5

*Sample* Prediction Results Using Alternate Models

| $S_a$ | $N_a$ | $D_a$' | $D_a(s)$ | $D_a(n)$ | $D_a(sn)$ |
|---|---|---|---|---|---|
| 264.39 | 142.17 | 5.87 | 5.09 | 9.09 | 7.66 |
| 312.24 | 132.62 | 7.32 | 6.52 | 8.46 | 7.76 |
| 167.08 | 83.88 | 3.00 | 3.05 | 5.26 | 3.92 |

In applying the predictors, we must recognize the three modes in which prediction is used: 1) metrics have been collected on some initial modules during design and we want to predict quality using these metrics; 2) initial metric data have been collected and we want to predict the effect on quality if changes are made in the design; and 3) we want to develop a software design strategy *prior* to writing the code and want to predict quality for *given* values of the metrics. We illustrate the last application. Suppose we want to compare predictions of quality for one design in which $S_a$=50 and $N_a$=20 is the goal and a second design in which $S_a$=100 and $N_a$=40 is the goal. Using $D_a(sn)$, we predict *avedrcount* of 1.84 and 2.39, respectively. In other words we predict a 100% increase in size will result in *approximately* a 30% degradation in quality. We must caution that the second design approach may be the more appropriate one, all things considered (e.g., requirements of the application). Also the smaller design would not be the better one if a decrease in module size increases *system* complexity as a result of increased inter module complexity of a large number of small modules. However, despite these caveats, the predictions allow software managers to assess *potential* quality problems.

## RESULTS AND CONCLUSIONS

- Using metrics to predict quality was successful. A next step is to see how well the approach fares with using a much smaller sample size for validation and subsequent application to a much larger universe for controlling and predicting quality, as would be the case in practice.

- Boolean OR functions of metrics were developed that act as effective discriminators of quality.

- It was found that using two metrics -- *statements* and *nodes* -- provided better discrimination than either alone. Additional research will be done to determine whether using additional metrics increases *Discriminative Power* even more.

- Because our experience with other metrics data had yielded much greater success in developing discriminators of quality as opposed to predictors of quality, such as regression equations, we were pleasantly surprised to be able to develop accurate non-linear regression equations of *drcount* as a function of one or more metrics (*stmts, nodes*) for use in quality prediction. We attribute this success to the use of data smoothing and the large sample that made smoothing feasible.

## ACKNOWLEDGEMENTS

We wish to acknowledge the support provided for this project by Dr. William Farr, Naval Surface Warfare Center; the data provided by Prof. John Munson of the University of West Florida; and the assistance provided by Mr. Ted Keller, Mr. David Hamilton, and Ms. Patti Thornton, International Business Machines Corporation.

## REFERENCES

[CON71] W. J. Conover, Practical Nonparametric Statistics, John Wiley & Sons, Inc., 1971.

[IEE93] Standard for a Software Quality Metrics Methodology, IEEE Std 1061-1992, March 12, 1993.

[JEN68] Gwilym M. Jenkins and Donald G. Watts, Spectral Analysis and its Applications, Holden-Day, 1968.

[JOB92] J. D. Jobson, Applied Multivariate Data Analysis, Volume II: Categorical and Multivariate Methods, Springer-Verlag, 1992.

[KHO92] Taghi M. Khoshgoftarr, J. C. Munson, B. B. Bhattacharya, and G. D. Richardson, "Predictive Modeling Techniques of Software Quality from Software Measures", Transactions on Software Engineering, Vol. 18, No. 11, November 1992, pp. 979-987.

[KLE78] David G. Kleinbaum and Lawrence L. Kupper, Applied Regression Analysis and Other Multivariate Methods, Duxbury Press, 1978.

[MUN93] John C. Munson and Ruth H. Ravenel, "Designing Reliable Software", Proceedings of the Fourth International Symposium on Software Reliability Engineering, November 3-6, 1993, pp. 45-54.

[SCH92a] Norman F. Schneidewind, "Methodology for Validating Software Metrics", IEEE Transactions on Software Engineering, Vol. 18, No. 5, May 1992, pp. 410-422.

[SCH92b] Norman F. Schneidewind, "Minimizing Risks in Applying Metrics on Multiple Projects", Proceedings of the Third International Symposium on Software Reliability Engineering", Raleigh, NC, October 9, 1992, pp. 173-182.

[SCH93] Norman F. Schneidewind, "Software Reliability Model with Optimal Selection of Failure Data", IEEE Transactions on Software Engineering, Vol. 19, No. 11, November 1993, pp. 1095-1104.

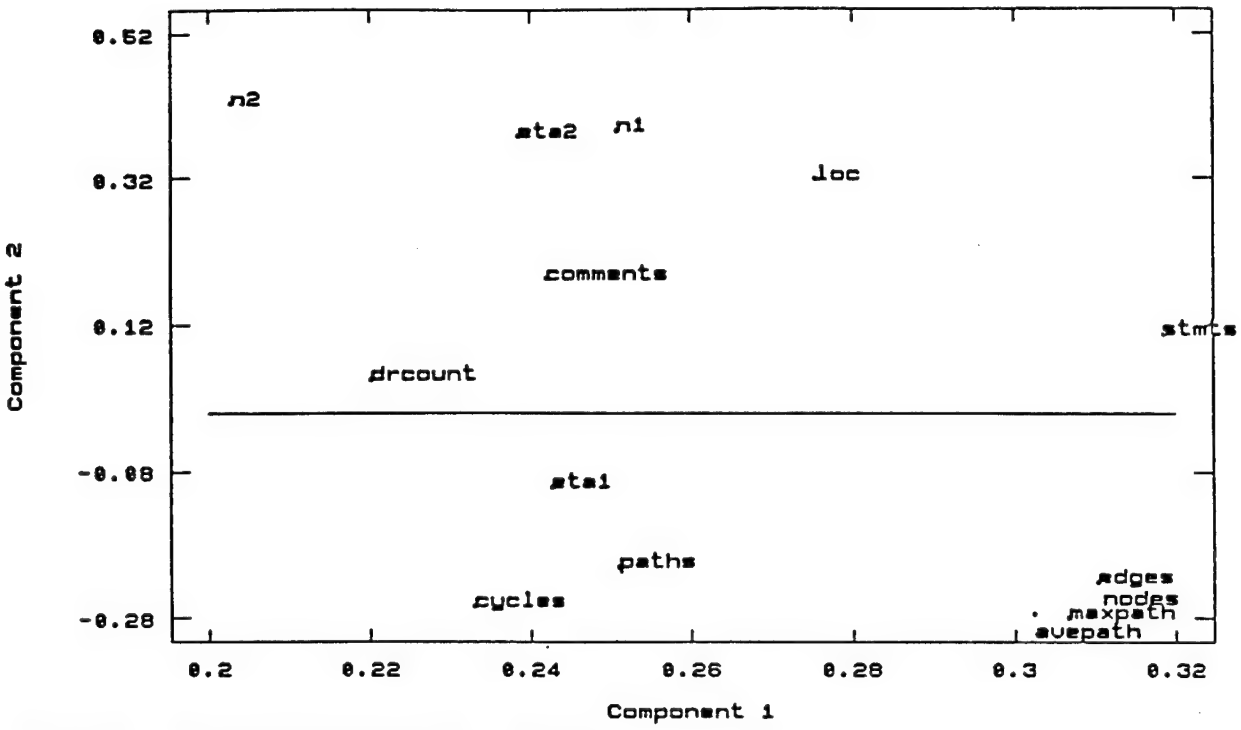



Figure 1 Principal Components Weights

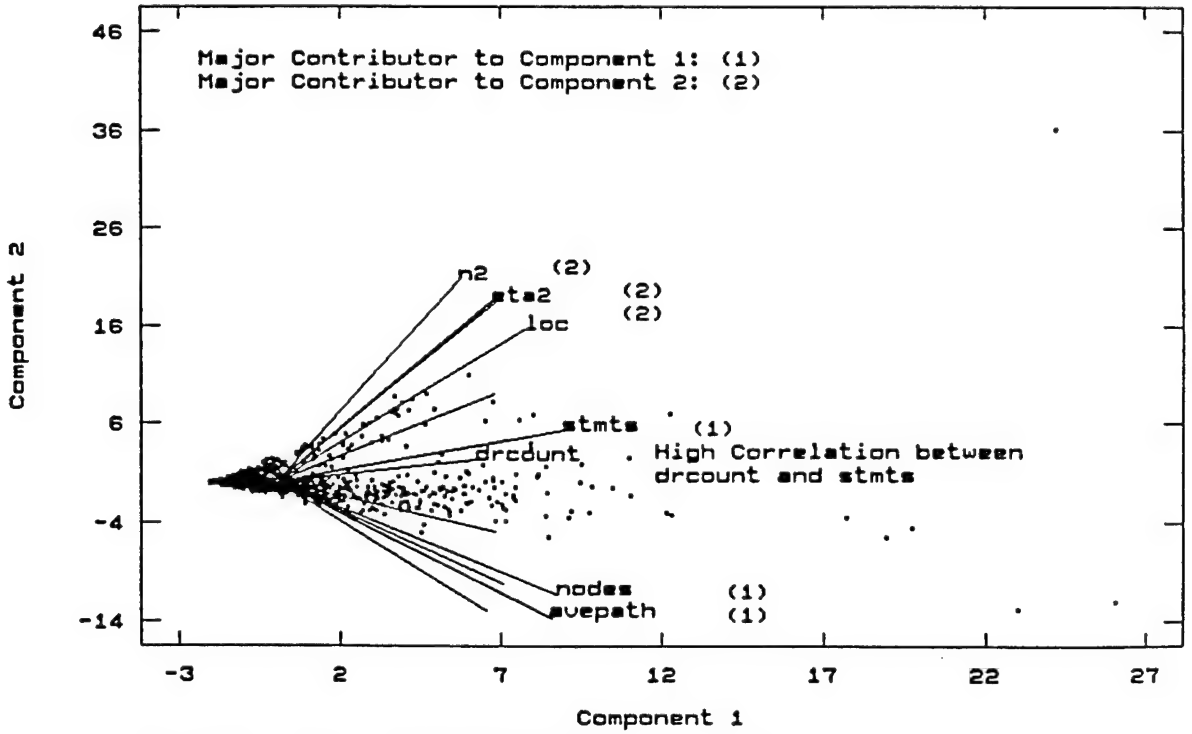



Figure 2 Major Contributors to Components

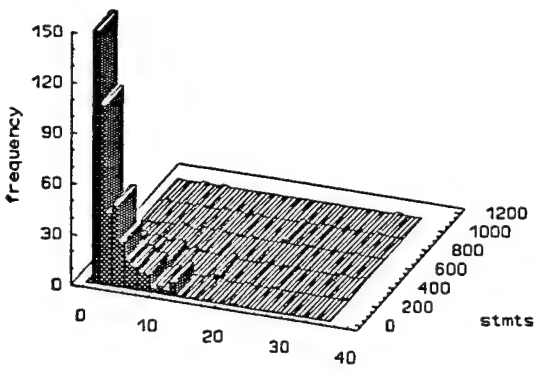



Figure 3 stmts and drcount histogram

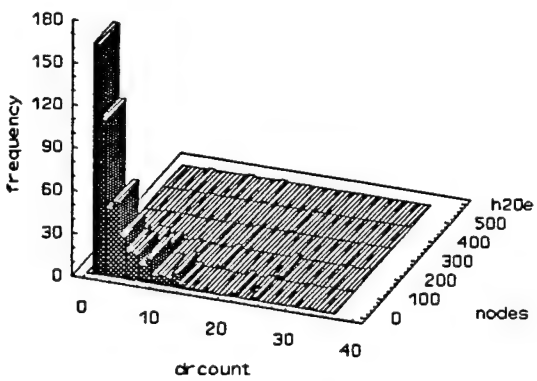



Figure 4 nodes and drcount histogram

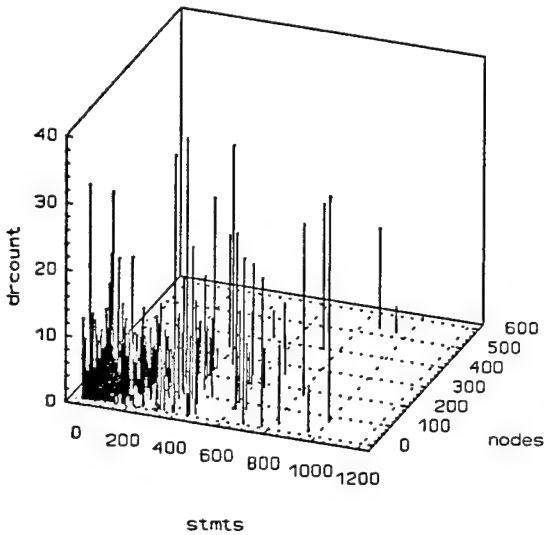



Figure 5 drcount versus stmts and nodes

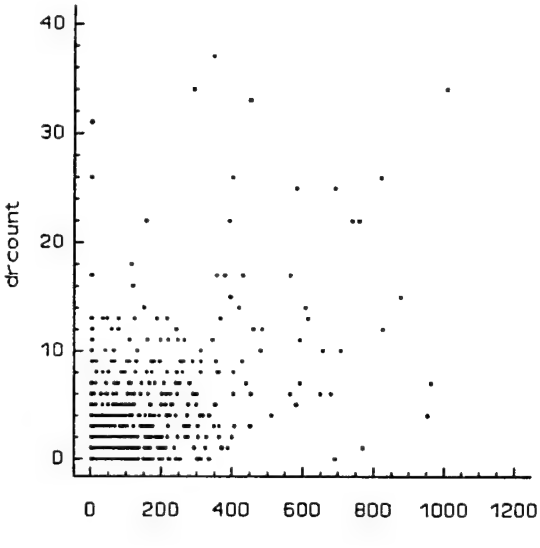



Figure 6 drcount versus stmts

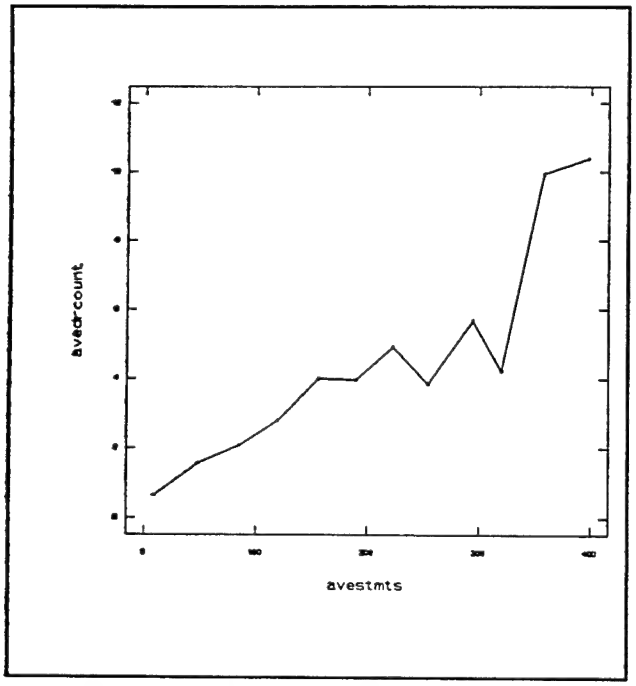



Figure 7 Observed avedrcount vs. Observed avestmts

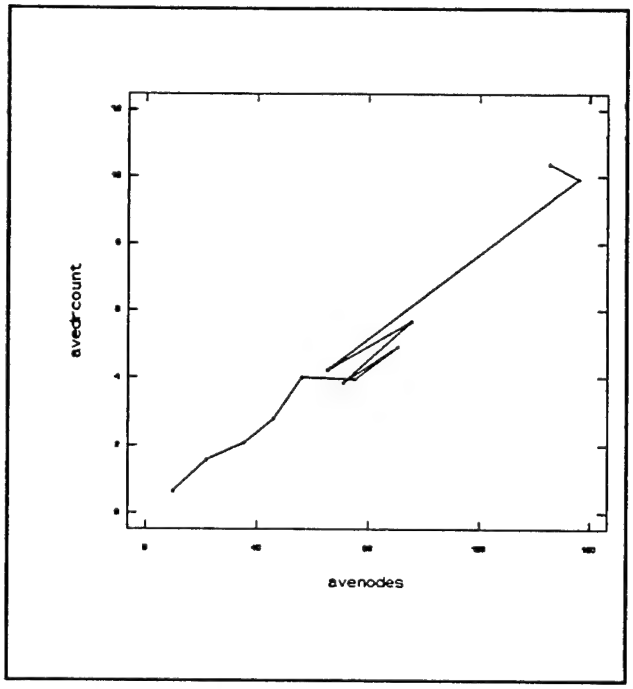



Figure 8 Observed avedrcount vs. Observed avenodes

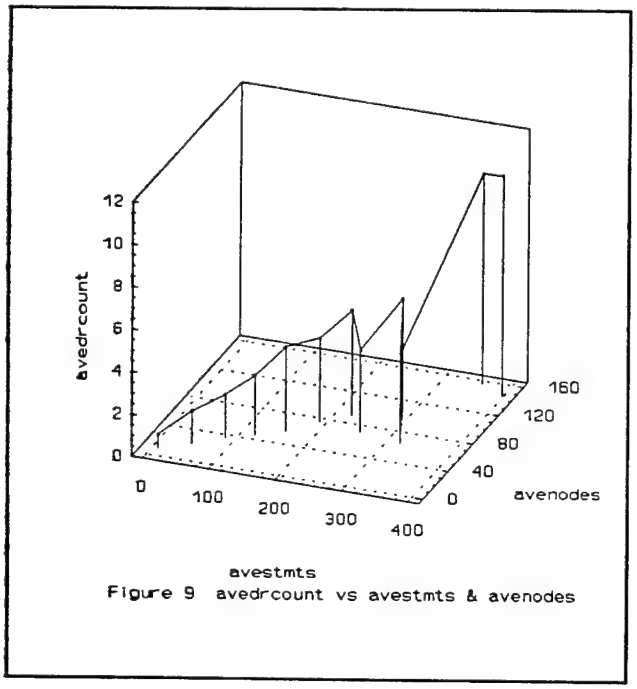



Figure 9 avedrcount vs avestmts & avenodes

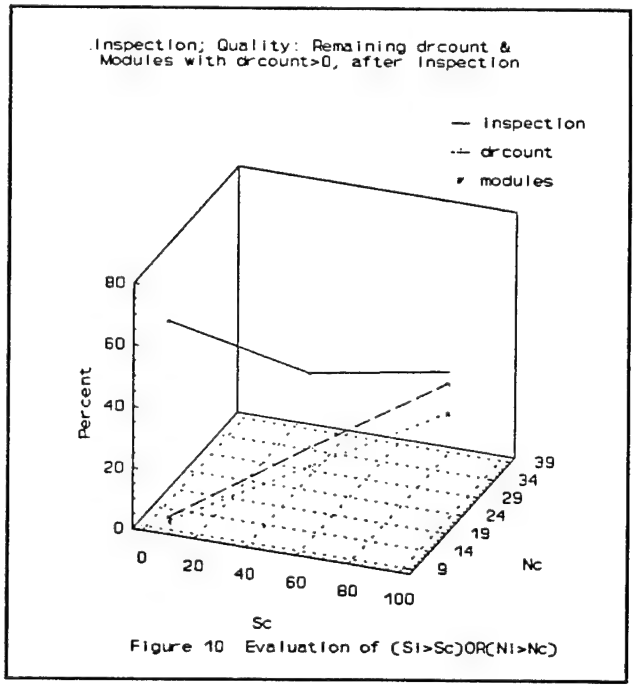



Figure 10 Evaluation of (SI>Sc)OR(NI>Nc)

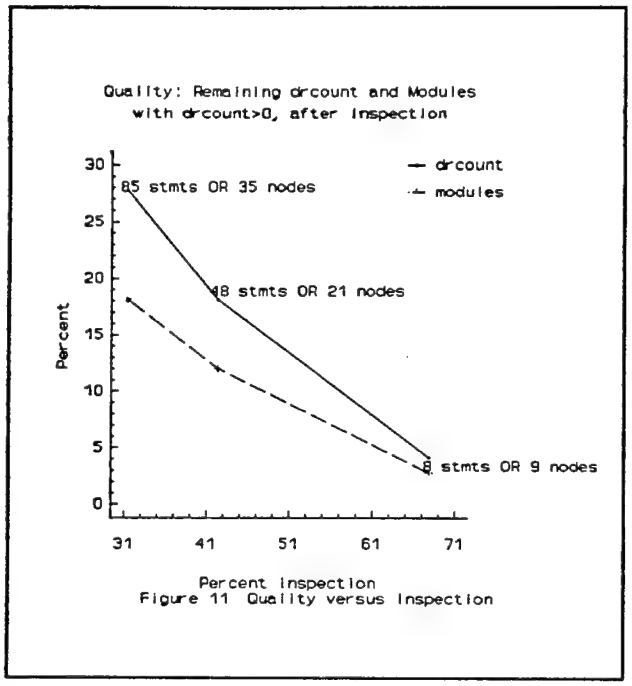


Figure 11 Quality versus Inspection

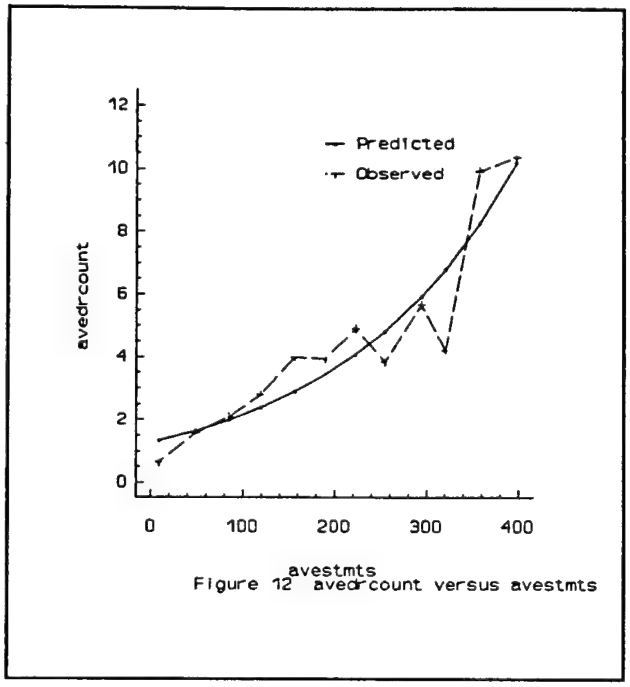


Figure 12 avedrcount versus avestmts

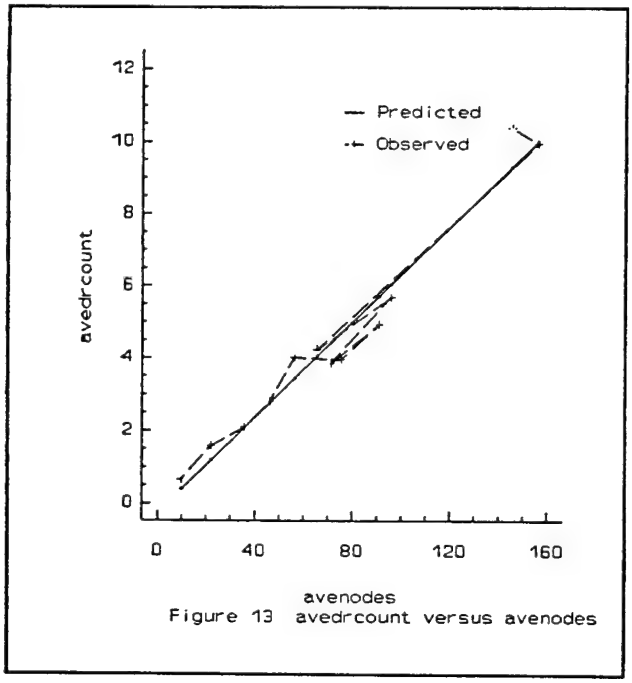


Figure 13 avedrcount versus avenodes

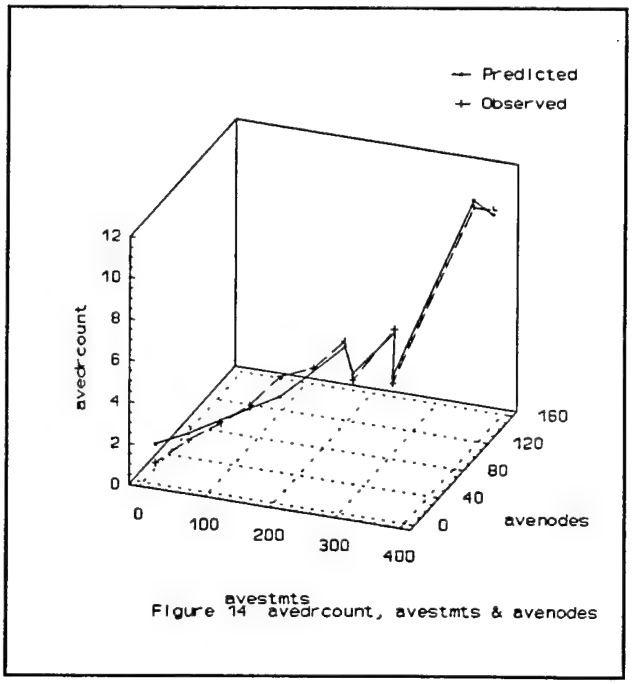


Figure 14 avedrcount, avestmts & avenodes

# MODEL-BASED SYNTHESIS OF COMPLEX EMBEDDED SYSTEMS

J. Sztipanovits, B. Abbott, T. Bapty, A. Misra
Department of Electrical and Computer Engineering
Vanderbilt University
Nashville, TN 37205

email: sztipaj@vuse.vanderbilt.edu

Abstract

This paper describes a model-based programming environment and program synthesis system, the Multigraph Architecture (MGA), which has been developed for large-scale embedded computer applications. In MGA, domain specific modeling paradigms are used to represent multiple views of the software, its environment, and their interrelation. The models are translated by model interpreters into executable programs that can run on parallel/distributed computing platforms. The model interpreters also generate input data for system's engineering tools that analyze selected characteristics of the models.

# MODEL-BASED SYNTHESIS OF COMPLEX EMBEDDED SYSTEMS

J. Sztipanovits, B. Abbott, T. Bapty, A. Misra
Vanderbilt University
Nashville, TN 37205

## Introduction

Model-based synthesis of complex embedded systems has been the focus of the research conducted by the Measurement and Computing Systems Laboratory of Vanderbilt University over the last decade. The result of this work, the Multigraph Architecture (MGA), provides framework and tools for the construction and maintenance of large-scale, embedded applications running on heterogeneous, distributed computing platforms. MGA applications have been extremely successful in several significant military and commercial projects in the aerospace and process industries.

First, the general characteristics of the technology and the application experiences will be summarized. This summary will be followed by the description of an application example The discussion will be concluded with an overview of the expected impact of the research.

## Multigraph Architecture

The role and significance of models and modeling in software engineering is well recognized. In a recent paper [1], David Harel described a software development method which utilizes models for representing the function and behavior of the software to be implemented. These models are suggested to be used for formal analysis and execution, providing tremendous help in building large-scale, reliable systems.

Our approach is closely related to the method described by Harel. However, in our view, a model-based programming environment is inherently domain specific; it adopts the concepts, relations, composition principles and constraints of a given domain. A generally accepted approach in modeling large-scale, embedded systems is to focus on specific aspects of *the software system* to be implemented. Although the relationship between the environment and the embedded software is an essential part of the overall complexity, this information is usually present in the design only indirectly, in the form of implicit assumptions and hidden attributes. It is desirable to make this relationship explicit and to extend the scope of modeling. The most profound consequence of this integrated modeling approach is the pressure to make the modeling paradigm domain (environment) specific due to the following reasons:

- In many complex applications, particularly in the area of embedded and reactive systems, system designers view software as an implementation method of certain functionalities (controllers, monitoring systems, etc.) which are integral part of a complex environment. The natural way to specify these functionalities is to use the concepts and notions of the domain, independently from the way of their implementation.

- Many disciplines use formalized or ad-hoc modeling paradigms for describing the function and behavior of systems, and for expressing design requirements. Process flow-sheets in the chemical industry or signal-flow graphs in electrical engineering are examples of application specific concepts and notations. It is usually inefficient (and maybe impossible) to re-cast existing, thoroughly understood knowledge into a different modeling formalism.

- Criteria for model consistency and completeness are frequently domain specific and can be better represented and verified in the "natural" modeling paradigm.

In complex domains, for instance in chemical process management, the modeling paradigm is extremely rich and the embedded applications (monitoring, diagnostic, control systems, embedded process simulations, etc.) are also very complex. The Multigraph Architecture (see Figure 1) provides a generic framework and tools for: (1) building and testing complex, multiple view models, (2) transforming the models into executable program and/or extracting from the models information for systems engineering tools, and (3) integrating complex applications on heterogeneous distributed computing platforms. It has three fundamental characteristics:

1. *Domain specific, multiple view modeling.* The software system, its environment, and their interrelations are represented by domain specific, multiple view models. The domain specific modeling paradigms include concepts, relationships, model composition principles, constraints and representation techniques that are accepted and used in the application domain.

2. *Model building, checking and verification.* A generic, Graphical Model Editor tool provides customizable modeling environment for domain experts. It enforces domain specific constraints during model building, uses domain specific graphical formalism and supports checking the model components against consistency and completeness criteria. Explicit representation of the constraints among modeling views allows the verification of models using system-wide consistency and completeness criteria. The model builder tool provides interfaces to model databases. In complex MGA applications, the Model Database is an object-oriented database system.

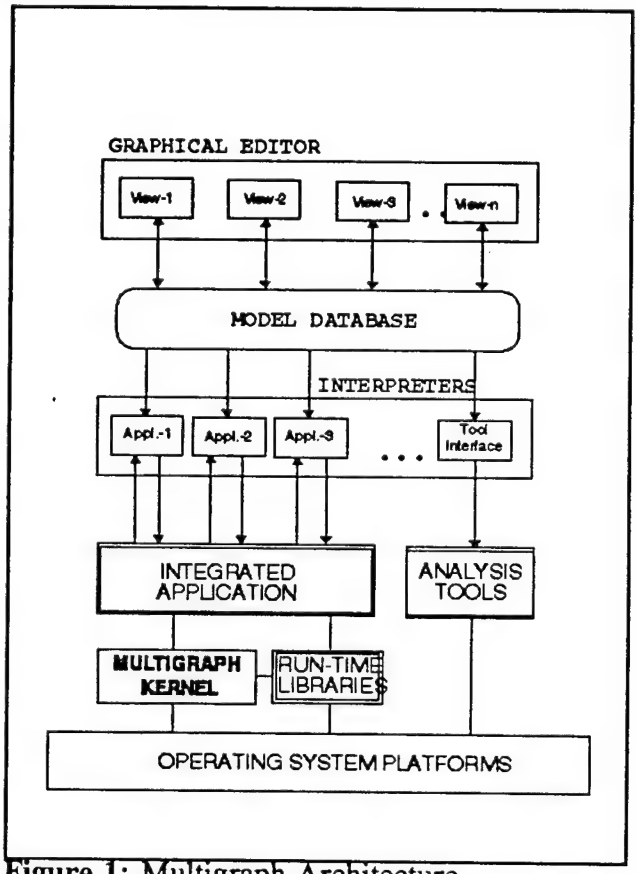


Figure 1: Multigraph Architecture

3. *Model interpreters.* The model interpreters transform the models into executable programs or generate data structures for systems engineering tools that perform various analyses of the system to be built. The model interpreters are specific to the applications to be generated. In general, a model interpreter traverses the Model Database, extracts the information which is relevant to the particular application and generates its executable. The executable programs are defined in terms of the Multigraph Computational Model (MCM). MCM is a macro-dataflow model, which provides a unified system integration layer above heterogeneous computing environments including open system platforms, high performance, parallel/distributed computers and signal processors. The run-time support of MCM is the Multigraph Kernel. A unique capability of the proposed technology is its *support of the reconfiguration of the executing system while it is running* [2] [3]. Model Interpreters are also used to generate data structures for systems engineering tools that perform various off-line analyses of the system to be built.

Currently, the construction of an MGA environment for a new domain includes the following steps:

Step 1. *Definition of the modeling paradigm.* This step requires understanding of the concepts, relations, model composition principles and constraints of the domain.

Step 2. *Customization of the model building tool and the model database.* Customization of the Graphical Model Builder tool requires the following specifications: (a) assignments between modeling concepts and graphic symbols, (b) assignments between graphic operations and model composition principles, (c) assignments between domain specific modeling constraints and admissible graphic operations. Customization of the model database requires the specification of database schemata (using an Object Definition Language).

Step 3. *Specification and implementation of model interpreters.* Model interpreters perform a mapping between "model objects" and "run-time objects". Model objects are selected, relevant components of the model which are accessible in the model database. Run-time objects are described in terms of the Multigraph Computational Model. Specification of this mapping depends on the type of the system to be synthesized, therefore different functional components of the executable system (e.g. "simulator", "diagnostic system", "signal processing system", etc.).

Step 4. *Implementation of core run-time libraries.* Run-time libraries typically consist of software modules of subroutine-size, with standard interfaces to the Multigraph Kernel. Due to their small size, individual modules can be easily implemented and tested. The resulting code is re-usable in different application domains.

In summary, MGA is a complete environment which integrates tools together for system modeling, system analysis, program synthesis, debugging, performance analysis and domain specific run-time libraries. The MGA has evolved as a programming and execution environment for large-scale embedded applications developed for instrumentation, process control, signal processing, and FDIR (fault detection, isolation and recovery) systems. In its current form, MGA provides an architectural framework for modeling, model representation, model interpretation and execution, and includes tools and run-time system components, which are customized to specific domains. Currently, MGA-based tools are supported on standalone and networked Unix workstations (Sun, HP9000, IBM6000, SG), PC's, and distributed memory multiprocessors (Transputers and networks of TI-C40-s).

## Example 1: Health Monitoring and Diagnosis for Aerospace Systems

The problem of constructing integrated health monitoring and diagnosis applications for large, complex aerospace systems has very large significance. High-performance computing is required for support of complex design analyses and operational diagnoses of large-scale vehicles. Applications in this area employ very large, high-fidelity models and must fuse high rate data from multiple sources into an architecture to support tasks in avionics design, failure effects analysis, and realtime failure detection, isolation and recovery.

MGA is the software framework of a model-based robust diagnostic system [4] and diagnosability/testability analysis tool, which is used by the Boeing Company on the International Space Station Alpha (ISSA) Program. MGA has been accepted by the ISSA Prime Contractor as a systems engineering tool. In this application, the multiple view modeling environment supports the *functional, structural, and behavioral* modeling of ISSA system components. Additional modeling views support the representation of fault detection activities, operator communication schemes. Tools of the modeling environment allow graphical model building, support extensive model consistency checking, and provide interface to related models, such as FMEA models available in engineering databases.

The system has several model interpreters. The model interpreter for the Diagnosability/testability Analysis Tool (DTool) [5] extracts relevant information from the multiple view models and synthesize data structures required by DTool. DTool evaluates detectability, distinguishability and predictability of faults given on-line sensor allocation and built-in-test (BIT) coverage, generates optimum test sequences, and provides advice for additional sensors/BIT coverage to meet defined criteria.

A different family of model interpreters automatically generates executable code of the real-time diagnostic system from the same model-set allowing significant savings in system/software engineering time. Components of the real-time fault detection/diagnostic system are synthesized in the Multigraph Execution Environment.

## Example 2: Parallel Instrumentation System

One of the major applications of MGA has been the development of model-based software technology for large-scale, real-time monitoring and signal processing systems running on heterogeneous parallel computing environments. In close cooperation with the USAF Arnold Engineering and Development Center, we have been developing the Computer Assisted Dynamic Data Monitoring and Analysis System (CADDMAS). CADDMAS provides real-time vibration analysis for 48 channels of 20 kHz bandwidth using a heterogeneous network of nearly 100 processors. Different versions of the CADDMAS are now applied as primary test systems in the turbine engine testing facilities of AEDC. A new version of CADDMAS will be installed in the Structures and Dynamics Laboratory of NASA-MSFC for Space Shuttle Main Engine testing.

In this application, the Modeling Environment supports the hierarchical modeling of the signal processing system, the hardware architecture and resource limitations. One of the model interpreters synthesizes the executable program for the parallel computing platform [3]. Several model interpreters generate input data for analyzer tools. For example, one of the interpreters generate a Stochastic Petri Net (SPN) specification of the executable system. This specification is evaluated by a SPN solver, which provides a performance evaluation of the system before its actual implementation.

## Conclusion

The model-based system synthesis approach offers several fundamental advantages in the design and implementation of complex systems.

1. The multiple view modeling environment captures design information, and allows the explicit representation of interdependencies among the different views.
2. Extension of the modeling views with additional views allows incremental design and helps to enforce consistency among the system components.
3. The models are active: model interpreters generate inputs for systems engineering tools to verify the design, and synthesize executable code in the system implementation phase.

A current research effort targets the introduction of formal methods in the specification of modeling paradigms and model interpretation procedures.

## References

[1] Harel, D.: "Biting the Silver Bullet," *IEEE Computer*, pp. 8-20, January, 1992.

[2] Sztipanovits, J., Wilkes, D., Karsai, G., Biegl, C., Lynd, L: "The Multigraph and Structural Adaptivity," *IEEE Transactions on Signal Processing*, Vol. 41, No. 8., pp. 2695-2716, 1993.

[3] Abbott, B., Bapty, T., Biegl, C., Karsai, G., Sztipanovits, J.: "Model-Based Software Synthesis," *IEEE Software*, pp. 42-53, May, 1993.

[4] Misra, A., Sztipanovits, J., Carnes, R.: "Robust Diagnostic System: Structural Redundancy Approach," in *Proc. of the SPIE's International Symposium on Knowledge-Based Artificial Intelligence Systems in Aerospace Systems in*

*Aerospace and Industry*, Orlando, FL, April 5-6, 1994.

[5] Misra, A., Sztipanovits, J., Underbrink, A., Carnes, R., Purves, B.:"Diagnosability of Dynamical Systems," *Proc. of the Third International Workshop on Principles of Diagnosis*, pp. 239-244, Rosario, WA 1992.

# Economics of Resource Allocation

Carlos C. Amaro
Alexander D. Stoyenko
Ami A. Silberman
Matthew Harelick
Thomas J. Marlowe
Nicola Jones
Tuna Tugcu
Purnendu Sinha
Phillip A. Laplante *
Bo-Chao Cheng

**Abstract**

We present an underlying conceptual model and recent developments in the DESTINATION project. Topics discussed include Cost Functions, Scaling & Data Fusion; evaluation of Constraints vs. Inputs with regard to System Design Factors; implementation of the Constraint Specification Interface; integration of the Logger; and development of the Workload Generator.

## 1 Introduction

Economics can be defined as *the science concerned with the problem of using or administering scarce resources so as to attain the greatest or maximum fulfillment of society's unlimited wants* [McConnell]. Similarly, complex applications consist of a set of logical modules to be allocated and executed on a distributed network according to the unlimited needs and wants (objectives) of the designer/user. In providing a given level of services we also aim to minimize cost which can be measured in terms of Money, Time, and Opportunity Cost, herein denoted M, T, and OC respectively.

The resource allocation problem deals with partitioning, allocation and scheduling of logical modules onto implementation nodes to provide an effective system design in a near-optimal manner by balancing the load evenly over all the available resources. The problem is similar in nature to the partitioning and scheduling problems considered for parallel program design [Lewis], [El-Rewini]. Assignment of processes to processors is facilitated by the Systems Engineering Technology Interface Specification (SETIS) file structure which includes a description of the Naval Surface Warfare Center's (NSWC) design framework Design Structuring and Allocation Optimization (DESTINATION) program and the network on which it is to execute. SETIS captures information for the Implementation model, for the Logical model, and for SDFs. The problem considered here involves numerous criteria including fault-tolerance, security and predictability, and/or performance of the resulting system.

In this submission we consider the problem of optimal assignment of tasks driven by a set of constraints, and the construction of the objective function which optimizes the performance. We discuss the evaluation of different types of constraints and the implementation of the constraint specification interface. We then discuss integration of the Logger, which provides a means to store user-generated information, with DESTINATION. Finally we treat development and implementation of the Workload Generator for creating arbitrary software and hardware models and an initial assigment relating them.

## 2 Background

We define a complex system to be a large application running on a distributed and heterogeneous network with a known but arbitrary topology, driven by various constraints such as performance, real-time behavior, and fault tolerance. These constraints frequently conflict, and satisfaction of these design objectives interacts

*Authors are with the Real-Time Computing Laboratory, Department of Computer & Information Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102 USA. E-mail: *dest@inis.njit.edu*. Laplante is also with the Department of Mathematics and Computer Science, Fairleigh Dickinson University, Madison, NJ 07940, E-mail: *laplante@fdumad.fdu.edu*. Marlowe is also with the Department of Mathematics and Computer Science, Seton Hall University, South Orange, NJ 07079, E-mail: *marlowe@cs.rutgers.edu*.

strongly with assignment of system tasks to processors in a distributed environment. DESTINATION provides an assignment module which can be used to optimize the system, as measured by the value of a weighted combination of objective cost functions. The logical model describes the functional and behavioral views of the system with emphasis on understanding of the system from a dynamic perspective. The implementation model includes the description of the hardware, software, and human resources which represent a particular form of the system under design. The design approach in task distribution is to treat absolute requirements as satisfiable constraints, and to combine the other requirements into a single objective function.

System design factors constitute the overall target of the optimization strategy, and include performance, real-time considerations, and computation processing requirements. Cost functions are used to choose among assignments. Some system design factors may have particular cost functions affected by the allocation strategy, by scheduling methodology, or by both. DESTINATION handles assignments at compile time since satisfactions of objectives and constraints are strongly dependent on assignment; scheduling may be either a compile-time or run-time activity. A list of sample system design factors and cost functions appears in Table 1.

Table 1: Sample Cost Function.

| SDF | Cost Function | Function to minimize | Affected |
|---|---|---|---|
| Performance | Load Balancing | Difference in processor load | T |
| | | Difference in link traffic | T |
| | Communication | Message transmission time | T |
| | Elapsed time | Finish time of sink | T OC |
| Real-Time | Deadline | Percent of deadlines missed | T OC |
| | | Worst case tardiness | T OC |
| Security | Risk | Difference between required and actual | OC |
| Fault-tolerance | Failure | Probability of lost messages | T OC |
| | | Expected number of restarts | T OC |
| Resources | Locality | Number of active physical sites | OC M |

# 3 Cost Function Issues, Scaling & Data Fusion

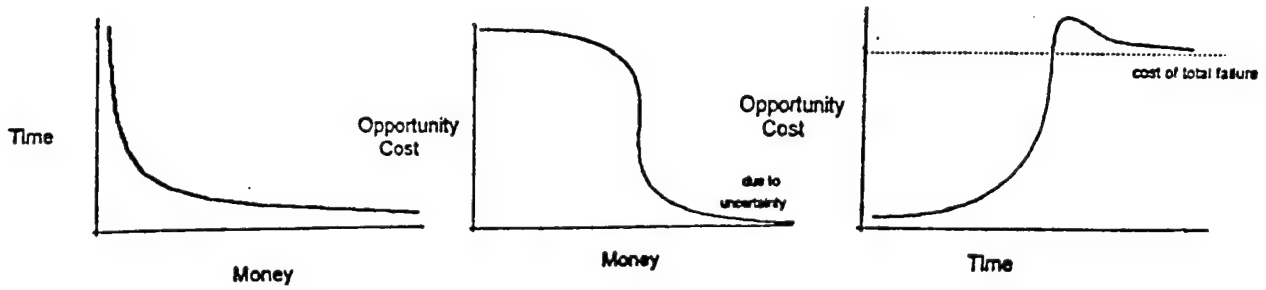
Cost functions evaluate system considerations used to determine the inherent "goodness" of an implementation. Among the various cost functions generally encountered in complex real-systems are communication cost, load balancing and elapsed time. We formulate cost functions so that minimization corresponds to better system performance, and minimizing the objective (cost) functions optimizes performance. The resulting objective function may combine many goals: performances measures such as communication or load balancing, human-factors, real-time, fault-tolerance, security, predictability, etc. At present, only communication and load-balancing have been implemented. These objectives which can be expressed in many diverse terms, such as time, money, resources, risk or opportunity cost, have fundamentally incompatible types and units. Meaningful combination of these objectives, measured in different units (sometimes artificial) and often of fundamentally different type, to obtain a single value and a valid solution requires Data Fusion.

In general, the system objectives can not be satisfied independently, but are correlated to each other. For example, the cost associated with the load balancing is inversely related to communication cost, and minimization of one tends to result in a large cost for the other. Again, the cost associated with minimizing elapsed time is directly related to a least laxity schedule. Simultaneous minimization of such interrelated functions will be hard and sometimes impossible. Measurements among objectives also need to be scaled to obtain approximately equal ranges and variances for different factors. Data scaling and data fusion are needed because:

1. The model is complex (arbitrary DAG with arbitrary messages of arbitrary cost.)
2. The goal is the optimal assignment of processes to processors (driven by a varying number of objectives and a set of constraints.)

Cost functions are combined in a linear combination using user-provided weights to reflect user and application priorities. After weights between zero and one have been specified for each of the objective functions,

Figure 1: Tradeoffs between Economic Factors




(a) T vs. M (b) OC vs. M (c) OC vs. T
(These are qualitative illustrations of relationships for particular scenarios.)

we can optimize accordingly. This function is then used to evaluate the cost of allocation strategies at each iteration of the optimization algorithms. The minimum cost iteration of an iteration set is saved as the new recommended allocation. A user may opt to save the current recommended allocation or continue with another iteration (At present, this is not implemented because the appropriate SETIS functions are not available.). The cost calculation is independent of the optimization strategy used: this allows the outcomes of different strategies to be compared. The various optimization algorithms currently incorporated in resource allocation are genetic algorithms, simulated annealing, neural network, and greedy algorithms like hill climbing.

Load balance is measured by the difference between the minimum and maximum utilization of processors. Communication cost is the total time actually used in sending messages. However collision and arbitration are not currently considered and need to be addressed in the future.

The execution time of a process on any processor is computed from the number of instructions of various types, and how long it takes to run those instructions on a specific processor. Currently, each process is considered to have a uniformly worst-case execution on all processors; we plan later to add explicit conditionals. The faster the processor, the less time is required. Communication cost is computed from the message length, link capacity, and protocol used. All this information is specified in the SETIS file.

Cost can be computed in dollar value of doing work, the time it takes to do it, or price to pay if something goes wrong. Each of these can easily be calculated individually. However, none of them are meaningful alone. Figure 1 shows some examples of tradeoffs.

The problem here is "to find a simple value that anyone can understand" which takes these three parameters into account: Money (monetary value of resources), Time, and Opportunity Cost (the true cost of choosing one alternative over another; that which is given up when a choice is made; i.e., robustness and reliability, degree of risk that things will go wrong). MTOC looks like part of the objective function group, indeed they are, but MTOC is on a micro scale where as objective functions are on a macro scale. Also, every objective function is directly related to and computed in terms of MTOC.

For example (Figure 1.a), given a computation-intensive task, we can get a general purpose computer to do the work, but it would take too long and increase the risk of something going wrong; we can alternatively use a supercomputer and get the work to be extremely fast and nearly correct but at a much higher price than we can afford.

Again (see Figure 1.b), if no money (resources) is spent to provide a solution to a problem, then all consequences (losses) due to the problem will occur. As resources are allocated, at first resources will make no difference due to the interdependence of problem set elements; only when sufficient resources are made available to fulfill most if not all of the problem requirements will there be a significant and nearly correct solution to the problem resulting in a significant reduction of OC. Due to uncertainty (unknowns: system failure, human error,

etc.), allocating more and more resources does not make any significant improvement after a while. Hence, resource allocation OC approaches zero but will not equal zero. It implies that there is always a better solution.

Finally (see Figure 1.c), if we had a machine that could do anything in zero time, then all solutions would be correct and timely, and opportunity cost among choices would be zero. Since, however, no resources can complete execution instantly, the time taken can never be zero; it may approach zero but will never reach it. If a problem has a good solution then the opportunity cost among those solutions is minimal but not zero, due to uncertainty. As the speed is lowered or workload is increased, the time to reach a solution increases which may not be acceptable and in terms increases the risk of failure. As more load is applied, critical components fail and the total consequences involve failure management plus the cost of producing results. Upon further reducing the speed of the machine/environment, and invalid results may be produced, and results are available less and less often so OC approaches the cost of the consequences of not producing a solution to the problem.

# 4 Evaluation of Constraints vs. Inputs to the Objective Function

The primary issue discussed in this section is how to identify constraints and differentiate them from system inputs. The outcome of this evaluation will be specification of a set of constraints to be displayed in the constraints window of the DESTINATION Project.

System Design Factors (SDF) specify system requirements for large systems. They allow allocation optimization algorithm objective functions to be tested. Inputs to the objective functions are SDF's, which are requirements for an entire system rather than specific elements of the system. Only system-wide requirements can be used for inputs to the objective function (although in the future guarded functions may allow inputs based on subsets). An example of an input is:

All tasks can expect to have an average throughput of $N$.

or, since these inputs will be used in optimization:

The total throughput of all tasks is to be maximized.

Constraints are specifications which:

- Describe conditions which *must hold* for single system elements.
- Describe a necessary relationship between a single system element and one or more system elements.

They may relate to single system design factors or multiple factors, or be independent of cost factors. Most constraints which do not relate to cost factors impose restrictions on assignments. A list of sample constraints appears in Table 2.

# 5 Implementation of the Constraint Specification Interface

There are two basic types of constraints. *Implicit constraints* are not expressible in the constraint graphical user interface; these constraints involve inherent limitations on processors (or links). For example, there can only be as many tasks on a single processor as there is memory space to accommodate them. These types of constraints are managed by the system automatically and should not be modified by the user.

*Explicit constraints* are constraints specified by the user. Explicit constraints are referenced by element type, such as a processor or a task. A constraint on the set of tasks $A$, $B$, $C$ is specified by referencing $A$ and then specifying constraints on its relationships with $B$ & $C$. The user can also reference $B$ or $C$ and specify constraints on the relationship with $A$. When specifying constraints on relationships between tasks and processors, the relationship will always be referenced by the task.

Table 2: Sample DESTINATION Constraints.

| SDF Constraints | |
|---|---|
| Type | Sample Constraint |
| Performance | Processor $p$ has memory $m$ |
| Real-Time | Task $t$ has absolute deadline $d$ |
| Fault-tolerance | Task $t$ must complete even if one processor fails |
| Reliability | The links of message $m$ have total failure probability at most $p$ during execution |
| | Task $t$ has failure probability at most $p$ during execution |
| Security | Message $M$ must use only secure links |
| Assignment Constraints | |
| Assignment | Task $t$ must be assigned to processor $p$ |
| | Task $t$ cannot be assigned to processor $p$ |
| Co-assignment | Tasks $t$ and $t'$ must be assigned to the same processor |
| | Tasks $t$ and $t'$ cannot be assigned to the same processor |
| Locality | Tasks $t$ and $t'$ must be located at the same physical site |
| Other | |
| Exclusion | Independent tasks $t$ and $t'$ may not co-execute |
| | Links $l$ and $l'$ cannot be in use simultaneously |

## 5.1 Interface Components

The main components of the Interface will be a Task Selection Window, a Processor Selection Window, and Result Window. A constraint is specified by selecting the reference element in either the Task or Processor Selection Window and then following a list of available options. The current constraints of a particular type will be shown in the Result Window.

## 5.2 The Task Selection Window

Once a task is selected, the user will be offered the options of Processor Relations, Task Relations, and Constraints.

The Processor Relation option will allow the user to specify constraints on relations between the selected task and one or more processors. One example of a constraint is "This task is allowed to exist on this processor." Once this constraint is selected, the results will appear in the Results Window in the form of an "Allowed Assignments" window. The user will have the option to toggle this to become a "Disallowed Assignments" window.

The Task Relations option allows the user to specify constraints on relations between the selected task and one or more other tasks. The Task Relations will provide further options on constraints such as Assignments and Real-Time. An example of an Assignment Constraint for Task Relations is "This task and task $B$ may not be assigned together on the same task." The Result Window for this option will be able to toggle between a "Allowed Co-assignment" and a "Disallowed Co-Assignment" list.

An example of a Real-Time Constraint for Task Relations is "The completion of this task must precede the completion of task $B$ by 9 seconds. " The Result Window will display the current task and the other tasks within this relationship ordered by time distance.

The Constraint Option allows the user to specify constraints on the selected task alone. Examples include Fault Tolerance constraints such as "This task may not fail", or "This task may be cloned", or the deadline for the task.

## 5.3 The Processor Selection Window

The Processor Selection Window allows the user to select among the options of Processor Relations and Constraints.

Processor Relations specify constraints on the relations between the selected processor and other processors. Some options for Processor Relations include Communications and Redundancy. An example of a constraint for communications is "This processor is allowed to send messages to processor $B$". The Result Window will toggle between a list of Allowed Processor Communication and Disallowed Communication.

An example of Redundancy is "If this processor fails than it can be replaced by processor $B$". There will be an appropriate toggleable Result Window display list.

The Constraints Option of the Processor Selection Window allows the user to specify constraints for this processor alone. An example of a processor constraint would be a throughput that is different than the throughput of the entire system.

# 6 Logger

The Logger provides a means to store information generated anywhere in the system, in whatever format the user requires. Each information entity generated by the user is stored as a time-stamped event. Events may be of any data type, including user-defined data types, and may be grouped in a user-defined configuration, so that for example, startup data can be separated from results. The Logger saves all data to files at user-definable intervals so that results can still be recovered should a system failure occur.

Events can be used to generate subsequent actions in the system by using the replay feature of the Logger. As a log is played back, either from the oldest event or the most recent, a user function is called for each event to generate new actions. Thus the Logger can be used to implement an automated test facility which could take the best results found so far and use them to produce further experiments. The Logger is fully configurable and is designed to be as flexible as possible.

# 7 Workload Generator

The Workload Generator is the principal part of the simulation and testing tool. For a given network topology, and a given problem size, the Workload Generator:

- Creates a number of software models of the given size (tasks and messages); and for each
- Assigns parameters to processors, links, tasks, and messages;
- Constructs a (random) initial assignment of tasks to processors (used as a seed by assignment search algorithms); and
- Checks this assignment for feasibility by examining it against constraints, and for optimization methods requiring an initial feasible assignment, generates another assignment and tests for feasibility.

Since it is possible for a user to specify an inconsistent set of constraints, only some fixed number of random assignments will be generated. The Initial Assignment Generator will eventually take account of assignment constraints in the generation phase. A later version of the Constraint Manager will include a limited consistency checker.

The Workload Generator makes fairly straightforward use of random number generators, optionally transformed to give a particular distribution (e.g., geometric or normal). The only moderately tricky issue is in efficient generation of the message edges, since the software graph must be a single-entry single-exit connected DAG.

# 8 Conclusion and Future Work

We have a running Resource Allocation component for the DESTINATION prototype at the Real-Time Computing Laboratory at NJIT. Current cost and objective functions include performance measures such as communication cost and load balancing. A number of assignment and optimization algorithms such as simulated annealing, genetic algorithm, neural networks, and hill climbing are operational within the prototype. Currently,

we are integrating fault-tolerance, reliability and security measures with the existing family of supported cost and objective functions. The Logger has been successfully implemented and tested as stand alone, however, work is in progress to integrate it with DESTINATION. The Workload Generator is in initial development stage.

The Constraint Interface will allow the user to explicitly determine constraints on and between individual system elements. The proposal thus far takes into account processors, links, tasks, message edges and relations among them. While other forms of devices are not addressed, it will not be difficult to add a Device Selection Window once constraints for them are identified. One example that can be identified immediately is Physical SDF's which can be used as constraints on individual devices. At this time, constraints do not have functionality in the system. Constraint functionality can be added through a Constraint Manager. In this scheme, the Constraint Interface will allow the user to graphically specify constraints which will be transformed into an algebraic/set theoretic specification of constraints. The Constraint Manager will then check the constraints for errors and consistency and provide requested information to the other parts of the system.

Managing scarce resources in order to efficiently fulfill the objectives is a complex chore. The complexity of this task is increased dramatically when the user is allowed to specify an arbitrary system and set of constraints. The existing system allows the user complete flexibility in the specification of a system allowing a wide range of constraints and objective functions providing various spectrum of MTOC tradeoffs. Since this system will allow the user to add whatever constraints and SDF's they desire, any type of complex system can be modeled; as we have shown with the various models of combinatorial optimization and assignment which we have implemented

# 9 Acknowledgements

We are indebted to generous support partially provided for this work under the U.S. ONR Grants N00014-92-J-1367 and N00014-93-1-1047, the U.S. NSWC Grants N60921-93-M-1912, N60921-93-M-3095, N60921-94-M-1426 and N60921-94-M-1250 and the AT&T UEDP Grant 91-134. We are also grateful for other DESTINATION team members, notably the CCCC company. Useful, constructive and valuable criticism and ideas have been provided by a good number of participants of the Software Synthesis Specification ad-hoc Working Group at NSWC, and by many Real-Time Computing Laboratory members and visitors.

# References

[1] Marlowe, T.J., Stoyenko, A.D., Laplante, P.A., Amaro, C.C., Nguyen, C.M., and Howell, S.L. "Multiple-Goal Objective Functions For Optimization of Task Assignment in Complex Computer Systems," IFAC/IFIP $19^{th}$ Workshop on Real-Time Programming, 1994.

[2] McConnell, C."Economics," $9^{th}$ ed. New York: McGraw-Hill, 1984.

[3] Lewis, T., and El-Rewini, H. "Introduction To Parallel Computing," Prentice-Hall, 1992.

[4] El-Rewini, H., Lewis, T., and Ali, H., "Task Scheduling in Parallel and Distributed Systems," Prentice-Hall, 1994.

# Application of Uncertainty Management System (UMS) Techniques to the Quantification of Confidence, or Belief, in Complex Systems

Tim Ramsey, Don Taylor and Russ Pimm

**Nichols Reseach Corporation**
4040 South Memorial Parkway
P.O. Box 400002
Huntsville, AL 35815-1502

## ABSTRACT

The most desirable method of establishing system performance is through the use of many test cases of actual hardware to compute statistical confidence measures. Complex systems are expensive to test and operational conditions may not be achievable under test conditions. For those complex systems in which the quantity and quality of test data is insufficient for statistical confidence estimation an approach for estimating confidence in overall system performance is required. This paper describes an Uncertainty Management System (UMS) approach based on tools and techniques currently in use in the UMS field. This approach will provide a tool for estimating the required uncertainties as well as support test planning and risk assessment.

## 1. INTRODUCTION

As systems become more complex it is increasingly difficult to determine their performance; consequently, confidence in estimation of performance has become a highly important consideration that is finding its way into user specifications of requirements. An example from a current Missile System Requirement is given below:

> *"Accuracy/Lethality requirements shall be demonstrated via validated 6-DOF computer/HWIL simulations to a 90 percent or greater confidence level."*

For relatively simple (and inexpensive) systems confidence in performance can be gained by field or flight testing or actual use in operational conditions. For complex systems it is not practical to establish confidence in this manner because of the costs involved. Thus, the testing of complex systems has come to rely on simulations or computer models to characterize their performance.

Using simulations to characterize (estimate) performance of complex systems is the main reason a user may ask the question (and specify a requirement):

> *"How can I be confident that the system will really perform as your estimates indicate?"*

The answer to this question cannot be obtained through purely statistical approaches to confidence level estimation. Statistical approaches deal primarily with the number of samples of performance taken, or the significance of a difference between two sets of data; but not with the validity of the samples themselves. The validity of the samples is what is now open to question, since they are obtained from simulations, rather than actual field tests.

Use of computer models and simulations to estimate performance of complex systems thus necessitates an expanded understanding of "confidence level" beyond purely statistical definitions, and a corresponding expansion of the means for numerically quantifying it.

The research described in this paper begins a strong rational approach to numerically quantify level of confidence in system performance. The approach makes use of tools and techniques from the emerging field of Uncertainty Management Systems (UMS) to expand the dimensions of confidence beyond its customary statistical connotations. This expanded treatment suggests the need for changing the terminology of the activity from "Confidence Assessment" to the more encompassing "Belief Assessment".

Our approach provides a mathematically and physically based technique for quantifying the belief level in performance estimates of complex systems. Having this capability provides a manager with the direly needed rationale for:

- Prioritizing program elements
- Estimating value of proposed tests
- Determining what tests (and measurements) are most needed
- Identifying high risk areas and how to reduce those risks.

## 2. OVERVIEW OF THE TECHNOLOGY

Our approach for quantifying belief level in performance estimates of complex systems makes use of a combination of existing performance simulations and newly developed tools from the field of UMS. Working in the domain of UMS allows a rigorous treatment of uncertainties which can then be transformed to "belief", or "confidence".

Uncertainties in the performance of complex systems arise from a number of sources, as shown in Figure 1. These uncertainties can be categorized into the four types of UMS uncertainties shown. Various activities are undertakenwithin a program that can reduce or eliminate these uncertainties. These activities represent sources of evidence as shown on the right side of Figure 1. The key problem, not yet satisfactorily solved for complex systems, is how to numerically incorporate evidence to calculate the resulting "belief" level in system performance.

Rigorous quantifications of uncertainty can be made using the tools of UMS[1]. Belief is the converse of uncertainty. An evaluation of four well known tools of UMS indicates that no single tool can provide all of what is needed for application to complex systems. Our approach combines the methods of Fuzzy Logic, Bayesian inference nets and Dempster-Shafer evidential reasoning into a synergistic system. The contributions of each tool to the approach are shown in Figure 2.

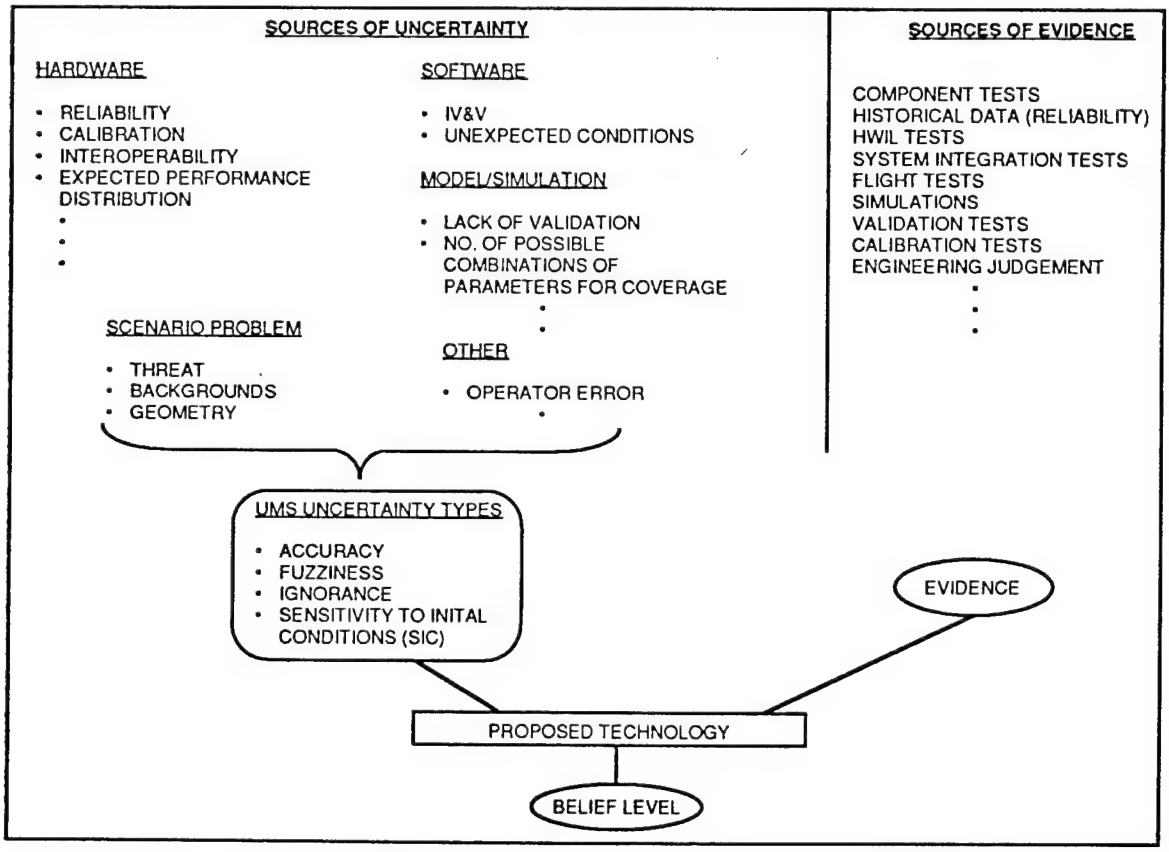



**Figure 1. Uncertainties in System Performance can be Reduced by Various Sources of Evidence.**

[1] *"Selecting an Uncertainty Management System"*, Peter Rothman, AI Expert, July 1989.

Bayesian inference nets provide a unique capability to structure knowledge from simulations in the form of conditional probabilities. They allow evidence about subsidiary system functions to be transformed into evidence for the parent function. These are important strengths but inference nets also have weaknesses. They provide no capability to quantify evidence. Errors in defining the *a priori* and conditional probabilities lead to errors in the conclusions. Most importantly, evidence that is not complete (unknowns remain) misleads Bayesian conclusions because the method has no means to quantify lack of knowledge.

The fuzzy logic notion of class membership values provides a natural method for quantifying evidence (degrees of membership in the classes of success and failure for the system function or model). Fuzzy logic also has good tools for combining evidence from multiple sources. However, it contains no natural way to quantify supporting evidence or the impact of lack of knowledge on belief. Neither does it provide a direct way of capturing probabilistic information.

Dempster-Shafer evidential reasoning provides explicit methods for quantifying lack of knowledge. It produces belief bounds that express the best knowledge of minimum belief supported by evidence and maximum belief limited by conflicting evidence. The distance between the bounds quantifies the remaining uncertainties. Dempster-Shafer does not have the ability to quantify evidence nor does it address the dependence of function success on related functions.

BAYESIAN INFERENCE (INFERENCE NETS)
- IMPLEMENTATION OF CLASSICAL CONDITIONAL PROBABILITY THEORY TO PROVIDE A STRUCTURE WHICH INCORPORATES DEPENDENCE OF FUNCTION SUCCESS ON PREDECESSORS
- PROPAGATES EVIDENCE FROM SUBSIDIARY FUNCTIONS AS INDIRECT SUPPORT TO HIGHER LEVELS
- PERFORMS WELL WHEN *A PRIORI* AND CONDITIONAL PROBABILITIES ARE KNOWN

FUZZY LOGIC
- EXTENSION OF SET THEORY TO COVER INHERENTLY POORLY DEFINED BOUNDARIES
- PROVIDES MEANS TO TRANSFORM TEST RESULTS TO EVIDENCE
- MEANS TO COMPARE DISSIMILAR INFORMATION FOR TRADE STUDIES (E.G., COST VS. CERTAINTY ENHANCEMENT)
- CAN PRODUCE SMOOTH DECISION CHANGES UNDER CONDITIONS WHERE OTHER SYSTEMS ARE TWITCHY

DEMPSTER-SHAFER EVIDENTIAL REASONING
- EXTENSION OF PROBABILITY THEORY TO MEASURES OF BELIEF
- MOST SOPHISTICATED HANDLING OF UNKNOWNS DURING EVIDENCE EVALUATION
- PRODUCES CONFIDENCE BOUNDS INCORPORATING BOTH SUPPORTING AND CONFLICTING EVIDENCE

**Figure 2. Three Well-Known Tools of UMS**

# 3. SYNERGISTIC BELIEF ESTIMATION

The belief estimation system, as currently implemented, consists of two applications. These are a specially configured relational data base and the belief computation software. Figure 3 illustrates the operation of a configuration to use test and simulation data to simultaneously estimate belief in achievement of desired performance and in simulation validity. In the Figure the solid lines indicate the flow of in formation from or about the system under development to the belief estimation process. The dashed lines represent the feedback of belief estimates.

Belief estimates for achievement of desired system performance levels consist of upper and lower belief boundaries. To define these boundaries test results are translated to evidence about specific system issues at the back end of the local data base. The minimum performance belief boundary is driven by evidence that supports the accomplishment of system goals. The maximum belief boundary is limited by the presence of evidence indicating that goals are not met. The degree of separation of the boundaries shows the residual uncertainties remaining.

Belief in the validity of models of functions of the system being developed are generated from a comparison of simulation and test results. To perform this comparison, the simulation data is interpreted to produce estimates of belief in performance achievement. These are compared with the belief boundaries created by examination of test data. If the belief estimate from simulation falls within the boundaries from test data, then the model is valid to within the test uncertainties. If the simulation belief falls outside the bounds, then action must be taken to resolve the inconsistency. The conclusion from this action should identify a model validity problem, a problem with the conduct

of a test or an error in interpretation of test results. In the latter case the interpretation problem can be corrected and new estimates made of belief in system performance.

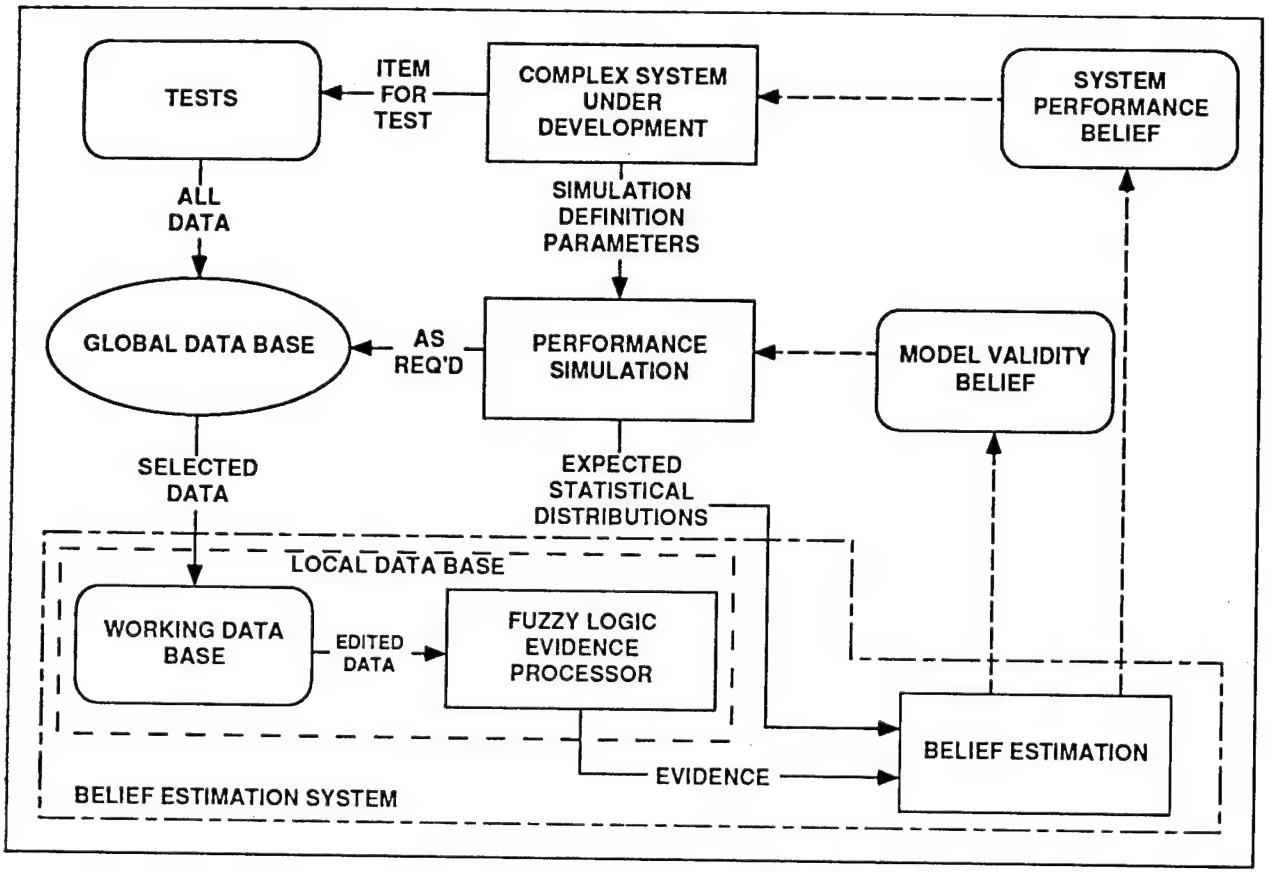



Figure 3. Operation of Belief Estimation System

## 4. APPLICATION TO COMPLEX SYSTEMS

Rigorous quantification of belief level has direct application to at service areas related to engineering of complex systems. These areas include:

Metrics technology to identify high risk.
Quantification of belief level is a metrics technology which by its very nature identifies components of requirements with high risk. Identification (and quantification) of risk (and how it can be reduced by various activities) is a key feature of the UMS technique. Establishing belief levels would be meaningless without dealing with risk.

Linking Requirements to Design.
Requirements on a system are usually expressed in terms of performance parameters that can be measured. The satisfaction of requirements is obtained by the system design - various components (hardware, software and "humanware") working together. Quantifying belief level in performance estimates requires the establishment of the effects of the various components on performance. Thus, a meaningful by-product of our approach will be the performance link between requirements parameters and the various design features of the system.

Specifying design goals/criteria.
A very important design goal which is often neglected is degree of risk involved. Requiring very long range from a sensor may seem like a good way to achieve overall system performance - until it is realized that developing a sensor with that capability involves untested technology and is therefore, very risky. A thorough belief-level analysis of system design and performance would uncover high risk areas early and allow for re-design of the system to alleviate the requirement, or for strong management attention on the high risk area.

Integrate design of high assurance into the system.

To be able to design the features of high assurance into a system it is necessary to know where the critical components are and on what they depend. The technique of belief estimation being proposed requires that these facets be identified, analyzed, and quantified.

# 5. SUMMARY

This paper summarizes work that is on going at NRC in the area of uncertainty management. The need for this capability is driven by increasingly complex systems which depend on simulations for a complete characterization of their performance. In combining the methods of several UMS software tools NRC is developing a capability that will not only permit quantitative assessment of the belief in a systems performance but will serve as a valuable tool in optimizing test programs by assessing the increase in belief in performance added by various test configurations. A tool such as UMS has been needed for some time and we believe it will in the future become a key tool in the evaluation process for complex systems.

# Integrating Cost Models with Systems Engineering Tools

**Thomas C. Choinski, (203) 440-5391**
**Daniel J. Organ, (203) 440-6546**

**Naval Undersea Warfare Center, Detachment New London, CT**

*Abstract*—**This paper considers cost modeling efforts within the Department of Defense over the last 30 years to formulate an approach for integrating cost modeling capability with the systems engineering tools developed under the Engineering of Complex Systems Block Program.**

## I. INTRODUCTION

The need to build complex warfare systems within an industry characterized by the declining defense budget has increased the role of cost modeling for defense systems. Accordingly, systems engineering design synthesis methodologies must incorporate cost modeling capabilities to insure robust tradeoff analyses.

Ideally, a sound systems engineering design synthesis methodology assesses the tradeoff between development cost, long term life cycle cost (excluding development cost), time, and functional performance. Figure 1 portrays this tradeoff.

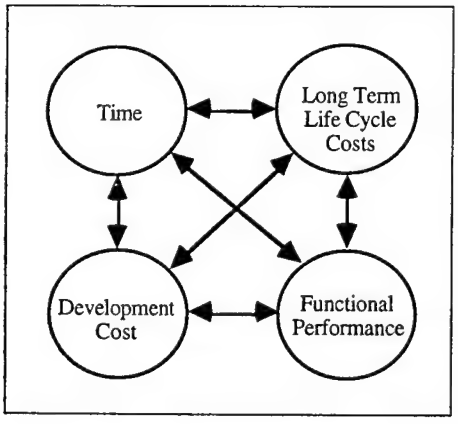



Fig. 1. Tradeoff analysis [1].

Realistically, the nature of projecting data for a warfare system platform that has a 30 year life cycle creates a significant variance on the data from any estimate. In the words of Paul G. Hough, one of RAND Corporation's premiere cost analysts: "cost analysis is subject to considerable uncertainty" [2].

The factors that contribute to the inexactness of cost estimates range from forecasting the cost/performance trends for new technology to projecting the rate of inflation for the life of the system. Short term estimates retain more accuracy than long term estimates; however, military systems require long term estimates. For these reasons, the tradeoff analysis often relies on qualitative data rather than quantitative data. The graph in figure 2 illustrates the difficulty historically encountered when forecasting the life cycle cost for complex military systems.

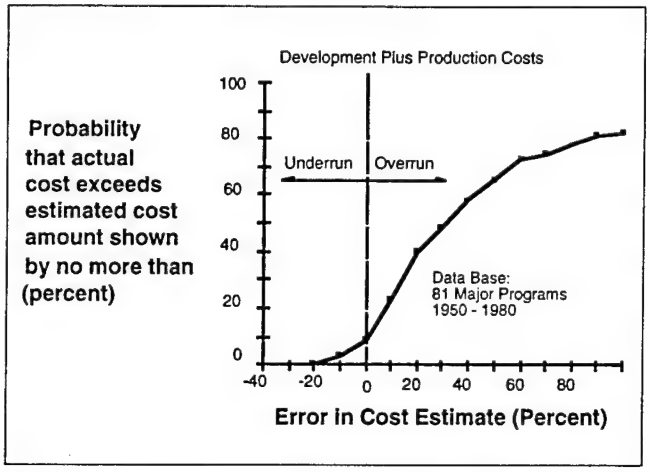



Fig. 2. Cost estimation track record [3].

This paper describes an approach for integrating cost modeling capability with the systems engineering tools developed by the Engineering of Complex Systems Block Program. The approach will:

- produce qualitative cost trends,
- incorporate detailed short term cost models,
- use heuristics for long term cost estimates, and

- incorporate existing models and tools, rather than develop new ones,

The goal is to enable systems engineers to perform qualitative cost tradeoffs while synthesizing complex engineering systems. Once the qualitative cost estimation capability has been incorporated into a systems engineering tool set, additional research can expand and refine the capability to provide more accurate cost estimation capability.

The discussion about the cost modeling integration approach advocated within this paper includes: a historical overview of cost modeling within the Department of Defense, a review of critical system design factors (SDFs) related to life cycle cost, an assessment of existing cost modeling tools, and a methodology for integrating cost models with the systems engineering tools developed under the Engineering of Complex Systems (ECS) Block Program. System Design Factors identify, and quantify important aspects of a system design [4]. The paper includes an extensive bibliography for the reader's benefit.

## II. Historical Overview

From its inception as a specialized field in the early 1950's to the present time, military cost analysis has evolved into an integral component of the decision making process for military system development. As cost analysis evolved, new methodologies were developed to improve the estimates.

In the 1950's cost analysis primarily evaluated cost estimates provided by contractors. Analysts compared the estimates against the cost of similar systems. This estimation methodology became known as "analogy" and will be described later in this section.

The 1960's was characterized by centralized decision making brought about by Secretary of Defense Robert McNamara and his "whiz kids". Cost analysis became an important aspect of the budgeting process as the nation struggled with funding the Vietnam War effort without adversely affecting the economy [2].

In the 1970's cost became a parameter of equal importance to performance. DoD Directive 5000.1 instituted the Design to Cost (DTC) concept. DTC led to the development of parametric model cost estimating. Life cycle costing (LCC) emerged when it became apparent that the existing models were deficient in estimating Operating and Maintenance (O&M) costs [2].

In the 1980's the Reagan administration instituted the Carlucci initiatives. The initiatives set out to improve the military procurement process. The legacy of the 1980's increased the media and public focus on cost analysis due to congressional hearings, legislation and high profile weapons systems.

Currently in the 1990's DoD has focused on developing affordable military systems. Cost has become a driving parameter. As a result the accuracy of cost estimates is critical; however, current state-of-the-art cost analysis does not, in most cases, provide better than a rough order of magnitude estimates [7]. Consequently, tradeoff analyses often rely on qualitative rather than quantitative estimates.

The importance of LCC, identified in the 70's, is reflected in the concept of viewing the life cycle and the associated costs in three distinct phases. These phases are research and development (R&D), investment (including production) and operation and maintenance (O&M).

R&D cost refers to all costs associated with research, development, test and evaluation of the system. This covers all costs during the validation and full scale development phase of a program. The costs associated with this phase end with the satisfactory completion of the initial operational test and government approval for use.

Investment cost refers to all costs associated with the production of the system. The costs incurred during this phase are complete when the operational system is delivered to the procuring command for use.

O&M cost refers to all cost associated with the operation and logistics support of the system subsequent to the delivery of the system.

DoD cost analysts focus on these three phases of the life cycle and their contribution to the overall life cycle cost. Cost related system design factors identify the components associated with each phase in greater detail.

## III. System Design Factors

System Design Factors provide a mechanism to quantify system characteristics for tradeoff analyses [4]. Tables I-III contain the SDFs for cost and their assignment to the three phases of the life cycle.

When considering SDFs related to cost, a systems engineer should concentrate on the most critical SDFs for each stage of the life cycle. Focusing on the SDFs that provide the most significant contribution to the cost at each stage of the life cycle, reduces the number of factors required for a total rough order of magnitude cost estimate.

For R&D cost the significant SDFs are: engineering, software, documentation, and test and evaluation. Engineering includes costs related to systems engineering and integration, design engineering, design support, and the redesign or formulation of engineering changes [5].

TABLE I
R&D SYSTEM DESIGN FACTORS [5]

| R&D | | |
|---|---|---|
| Validation | Engineering | Software |
| Program Management | Prototype Hardware | Support & Test Equipment |
| Documentation | Test Spares | Test Facilities |
| Test Personnel Training | Test & Evaluation | |

TABLE II
INVESTMENT SYSTEM DESIGN FACTORS [5]

| Investment | | |
|---|---|---|
| Production Hardware | Program Management | Training |
| Integration & Test | Production Test and Evaluation | Documentation |
| Production Support & Services | Industrial Facilities | Initial Spares |
| Installation and Checkout | Support & Test Equipment | Transportation |
| Operational Sites | Maintenance Sites | Supply Introduction |

TABLE III
O&M SYSTEM DESIGN FACTORS [5]

| O&M | | |
|---|---|---|
| Crew | Facilities | Material |
| Personnel | Modernization | Overhaul |
| Software Maintenance | Preventative Maintenance | Corrective Maintenance |
| Operator Training | Support and Test Equipment Maintenance | Replenishment Spares |
| Inventory Storage | Supply System Management | Repair Material |
| Documentation Maintenance | Shop Space | Shipping |
| Packaging Material | Material Handling Labor | |

For Investment cost (includes production) the significant SDFs are: production hardware, production support and services, initial spares, operational sites, and maintenance sites.

For Operating and Maintenance cost the significant SDFs are: crew, material, preventative maintenance, corrective maintenance, and modernization.

The aforementioned cost related system design factors become the focus of cost estimation efforts for a warfare system. Analysts have developed several techniques to estimate the components of life cycle cost for military systems.

## IV. Cost Model Assessment

Presently DoD cost analysts use four basic cost estimation methods [6]. These methods are: analogy, expert judgment, bottom up (Industrial Engineering), and top down or parametric estimation. Table IV presents a comparison of the four methods.

TABLE IV
METHODOLOGY COMPARISON

| Method | Accuracy | Time | Historical Dependence |
|---|---|---|---|
| **Analogy** | Low | Medium | High |
| **Expert Judgment** | Low | Low | High |
| **Bottom Up** | Medium | High | Medium |
| **Top Down** | Medium | Low | Low |

Analogy involves the comparison of similar systems. The analyst compares attributes of similar systems to determine a reasonable cost estimate. This approach requires a fair amount of experience, as well as historical data. The estimate is highly subjective to the bias and experience of the analyst. The method assumes the use of similar technologies. Nevertheless, the estimate can be generated quickly.

Expert Judgment is similar to analogy because the estimate can be generated quickly. In this approach, an "expert" uses his experience to generate a cost estimate. As with analogy, the estimate is sensitive to inaccuracies due to the subjective nature of the expert's opinion. This method depends highly on the availability and skill of experts.

The Bottom Up or Industrial Engineering method represents a more objective approach. This method divides the estimation task into smaller units. An individual or group who has the appropriate data, experience and model to generate an accurate result produces the estimate. The

sum of the unit costs determines the total cost. This approach yields a more accurate estimate, but it is time consuming. The availability of unit cost data determines the accuracy of the Bottom Up approach.

Top Down or Parametric Estimation is the most widely used cost estimation method within DoD [6]. This method relies on computer models to obtain the cost estimate. The analyst provides lower level design parameters as input to the model. Generally these models are easy to use, and produce a more accurate result. The difficulty with using parametric models is that occasionally the required input parameters may not be known or may be difficult to quantify. The lower level nature of some of the parameters often means they are difficult to define until the design progresses.

DoD uses parametric models primarily to estimate software costs. One reason for this is that software cost is a significant component of the system life cycle cost. In addition it appears that software cost may be more difficult to determine using the other estimation methodologies.

A variety of software cost estimation models exist. Many are proprietary. The more widely used models include [6]:

- Constructive Cost Model (COCOMO)
- PRICE-S
- System Evaluation and Estimation of Resources (SEER)
- Software Life Cycle Cost Model (SLIM)
- Revised Enhanced Version of Intermediate COCOMO (REVIC)
- Software Architecture, Sizing and Estimating Tool (SASET)

The models require the user to provide specific parameters that quantify and describe their system. The parameters required by the models cover: lines of code, language, number of programmers, experience of programmers, development platform, availability of tools, reliability and schedule. Analysts may have difficulty quantifying some parameters requested, such as developer experience. However, these models provide an estimate rather quickly. In addition, the models are generic, that is the software models can be applied to estimate costs for any application. System engineers using these models should understand that these models often are difficult to calibrate [7], and require recalibration for different environments [8].

While many generic models exist for software cost estimation, the same is not true for other aspects of the life cycle cost. These costs are generally computed by analogy, expert opinion or by models developed for a specific application. The models use heuristic data bases from existing combat systems to estimate cost. For example at the Naval Undersea Warfare Center (NUWC), several combat system specific cost estimation models exist based on data from existing submarine warfare systems.

The current state-of-the-art in cost estimation technology relies on generic models to estimate software costs and expert judgment/heuristic models for other components of the life cycle cost. Therefore, the integration of cost models with systems engineering tools should be flexible to incorporate the different types of models available for each phase of the life cycle.

## V. Integrating Cost Models

The integration of cost modeling capability into systems engineering tools will enable systems engineers to perform qualitative cost tradeoffs while synthesizing complex engineering systems. The objectives of the cost model integration process encompass:

- defining cost related systems design factors,
- pinpointing critical cost related system design factors,
- providing systems engineers with qualitative cost factors for systems design synthesis,
- linking cost related system design factors to the Design Structuring and Allocation Optimization (DESTINATION) systems engineering tool, and
- permitting design factor optimization within DESTINATION.

Existing cost models will be used in order to expedite the availability of cost modeling capability. Figure 3 depicts the method which will integrate cost models with the DESTINATION systems engineering tool developed by the Engineering of Complex Systems Program.

The method breaks down the cost models into three different categories: hardware research and development, software research and development, and the remaining life cycle costs. The hardware and software research and development cost models are kept separate because detailed cost databases exist for hardware, and established models exist for software (refer to the *Cost Model Assessment*

section of this paper for examples of existing databases and models). Heuristics will provide information for the remaining part of a system's life cycle. Heuristics are readily available and produce satisfactory information for this stage of the cost model integration process.

The link to DESTINATION will occur through the Systems Engineering Technology Interface Specification (SETIS) software. SETIS interface software will be developed to provide a link to the cost models. The SETIS link ensures compatibility with other tools developed within the Engineering of Complex Systems Program. System design factors describing the hardware resources, software design, functional requirements and programmatic requirements will be extracted from DESTINATION through SETIS. Additional, lower level, SDFs will be defined during the model integration process. The cost model will supply information regarding cost related system design factors to DESTINATION through SETIS.

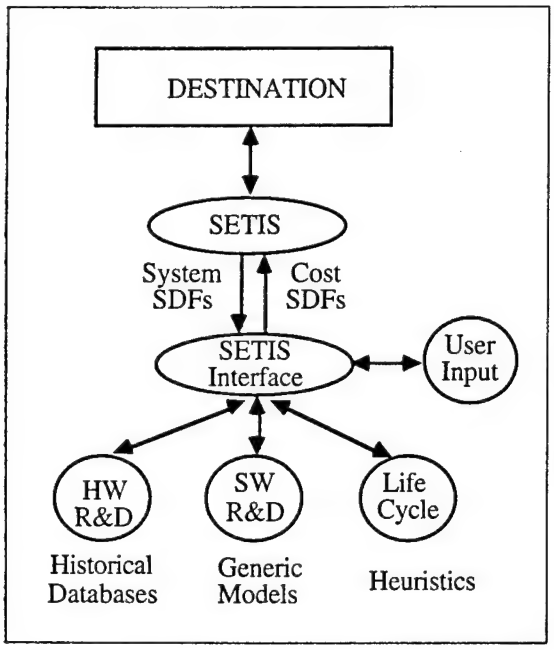



Fig. 3. Integration with DESTINATION.

If the system specific information currently provided by DESTINATION is inadequate, then the user must provide additional information to the cost models. The SETIS interface will also query the user to input the additional information and incorporate the data into the cost estimate.

The SETIS interface will manage the data generated from the hardware R&D, software R&D, and life cycle cost models. The interface software will require the development of C++ code to maintain compatibility with SETIS. SETIS was implemented using a C++ class hierarchy [9]. The intent is to develop cost modeling capability that requires minimum modification to existing SETIS software.

The cost related system design factors described in this paper will serve as the foundation for the cost models. The system design factors listed in *NAVSWC Technical Report 92-268* will also receive consideration. These system design factors include cost to: develop, prototype, produce, test, purchase, operate, maintain, repair, include security capability, and achieve productivity.

Once the concept has been demonstrated, the repertoire of cost models can be expanded to give the systems engineer different options and perspectives. Each new model would be linked to DESTINATION through SETIS, and the SETIS interface software developed for the cost models.

In addition to the integration of cost modeling capability with DESTINATION, the approach outlined can also be applied to the integration of commercially available system engineering tools. By using the CASE Document Interchange Format (CDIF) instead of SETIS, the cost estimation models can be integrated with system engineering tools such as RDD-100, Cadre Teamwork and others. Of course this approach assumes that these tools comply to the standardization of CDIF. If the selected tools don't comply with CDIF, some translation between specific formats may be required to interchange data between tools.

## VI. SUMMARY

This paper describes an approach for integrating cost modeling capability with the systems engineering tools developed by the Engineering of Complex Systems Block Program. The approach entails:

- using existing hardware R&D cost databases,
- incorporating established software cost models,
- estimating long term life cycle cost with heuristics,
- defining cost related system design factors, and
- designing a software interface to DESTINATION via SETIS.

The cost modeling integration approach was conceived by tracing through the history of cost modeling within the Department of Defense, defining cost related system design factors, assessing existing cost modeling

tools, and formulating a methodology for integrating cost models with the systems engineering tools developed under the Engineering of Complex Systems Block Program.

A qualitative cost estimation capability will enable system engineers to perform tradeoffs while synthesizing complex engineering systems. The fundamental tradeoff between development cost, long term life cycle cost, time and performance is the primary objective.

Finally, the cost modeling research that led to the proposed cost modeling approach produced an extensive list of references. Although many of these references were not explicitly used within the context of this paper, the full list is included for the reader's benefit.

## ACKNOWLEDGMENT

The authors would like to thank Dr. José Muñoz for providing the technical review of this paper.

## REFERENCES

[1] Thomas C. Choinski, "Economical Development of Complex Computer Systems," *Proceedings from the Complex Systems Engineering Synthesis and Assessment Technology Workshop*, Washington, D.C., 20 July 1992.

[2] Paul G., Hough, *Birth of a Profession: Four Decades of Military Cost Analysis*, The Rand Corporation, August 1989.

[3] Norman R. Augustine, *Augustine's Laws*, American Institute of Aeronautics and Astronautics, 1983.

[4] C.M. Nguyen and S. Howell, 'System Design Factors: The Essential Ingredients of System Design", *NAVSWC TR 92-268.*

[5] Naval Weapons Engineering Support Activity, Management Engineering Department, Cost Management Division, *Life Cycle Cost Guide for Major Weapon Systems,* November 1977.

[6] Brent L., Barber, *Investigative Search of Quality Historical Software Support Cost Data and Software Support Cost Related Data*, Thesis, Department of the Air Force, Air Force Institute of Technology, December 1991.

[7] Gerald L. Ourada, *Software Cost Estimating Models: A Calibration, Validation, and Comparison*, Thesis, Department of the Air Force, Air Force Institute of Technology, December 1991.

[8] Chris F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM*, Volume 30, Number 5, May 1987.

[9] Baba Prasad, et al., The System Engineering Technology Interface Specification (SETIS): An Update, *Proceedings from the Complex Systems Engineering Synthesis and Assessment Technology Workshop*, Washington, D.C., 20 July 1993.

## BIBLIOGRAPHY

American Power Jet Co. *ILS Element E15 Cost Analysis and Funding*, April 1991.

American Power Jet Co. *Structured Analysis, ILS Review Element E15 Cost Analysis and Funding*, June 1989.

Assistant Secretary of Defense (Systems Analysis). *Proceedings of the Annual Department of Defense Cost Research Symposium,* Volume I, March 1968.

Assistant Secretary of Defense (Systems Analysis). *Results of Department of Defense Cost Research Survey*, March 1970.

Baldwin, R.W. and D. E Emery. *Technology Assessment of the Software Life Cycle Support Environment,* Mitre Corp, August 1991.

Barbour, A. A. *An Improved NATO Military Force Cost Model (Namilfo)*, The Rand Corp., July 1969.

Batt, G. T. *Case Technology and the Systems Development Life Cycle: A Proposed Integration of Case Tools with DoD-STD-2167A*, Thesis, Naval Postgraduate School, March 1989.

Berkowitz, Prof. Murray R. *Military Computer Hardware, Software and Personnel Resources: A Critical Analysis,* Graduate School Of Management, The University of Dallas, 1982.

Bersoff, Edward H. and Alan M. Davis. "Impacts of Life Cycle Models on Software Configuration Management," *Communications of the ACM*, Vol. 34, No. 8, p. 105, August 1991.

Bozek, T. *Automated Information System Life-Cycle Management Manual*, Assistant Secretary of Defense (Comptroller), March 1990.

Bruckner, Blaine R. and K. J. Merrill. "Computer Integration of SEAWOLF Class Submarine Life-Cycle Functions," *Journal of Ship Production*, Vol. 7, No. 1, February 1991.

Collette, M. A. *Inquiry into the Cost of Post Deployment Software Support (PDSS)*, Thesis, Air Force Inst. of Tech., Wright-Patterson AFB, September 1989.

Cornyn et al. "Life Cycle Cost Models for Comparing Computer Family Architectures," *Proceedings from the National Computer Conference*, 1977.

Department of Defense. *Casebook of Life Cycle Costing in Equipment Procurement*, July 1970.

Department of Defense. *Life Cycle Costing Procurement Guide (Interim)*, July 1970.

Department of Defense. *Military Standard Systems Engineering (Draft),* MIL-STD-499B, 6 May 1992.

Department of Defense, *Military Standard for Design to Cost*, MIL-STD-337.

Department of Defense. *Military Handbook for Design to Cost,* MIL-STD HDBK-766.

Department of Defense. *Military Handbook for Cost Engineering: Policy and Procedures*, MIL-HDBK-1010A.

Department of Defense. *Military Handbook for Life Cycle Cost in Navy Acquisitions*, MIL-HDBK-259.

Fisher, Gene E. *Cost Considerations in Systems Analysis*, A Report Prepared for the Office for the Assistant Secretary of Defense (Systems Analysis), The Rand Corporation, December 1970.

General Accounting Office. *Navy Acquisition: Cost, Schedule, and Performance of New Submarine Combat Systems*, January 1990.

Graver et al. *Cost Reporting Elements and Activity Cost Trade-Offs for Defense System Software,* Volume I, Study Results, General Research Corp., May 1977

Graver et al. *Cost Reporting Elements and Activity Cost Trade-Offs for Defense System Software,* Volume II, Executive Summary, General Research Corp., May 1977

Green, Robert. *Defense Task Force to Clarify Life Cycle Guidelines,* Government Computer News, Vol. 10, No. 13, p. 49, June 24, 1991.

Greve, A. R. et al. *REVIC Advisor (REVAD): An Expert System Preprocessor to a Parametric Software Cost Estimating Model*, Defense Logistics Agency, Operations Research and Economic Analysis Office, September 1991

Gulezian, R.C. *Application of Statistical Methods to the Development of Naval Software Maintenance and Related Cost Estimation Models, Data Base Services*, May 1988

Hildebrandt, Gregory G. and Man-bing Sze. *An Estimation of UASF Aircraft Operating and Support Cost Relations*, The Rand Corporation, May 1990.

Johnson, M. Demarco. "Life Cycle Management: It's Already Broken," *Journal of Systems Management,* Vol. 42, No. 2, p. 32, February 1991

Jones, P. "Naval Life Cycle Costing - Still a Black Art to Industry ?," *Conference Proceedings MILCOMP 89, Military Computers Systems and Software*, p. 255-60, September 1989.

Kankey, R.D. *Challenge of Software Maintenance Costing*, Air Force Inst. of Tech., Wright-Patterson AFB, March 1989.

Kemerer, Chris F., and Rajiv D. Banker. "Scale Economies in New Software Development," *IEEE Transactions on Software Engineering*, Volume 15, Number 10, October 1989.

Kemerer, Chris F. "Bridging the Gap between Research and Practice in Software Engineering management: Reflections on the Staffing Factors Paradox," *Proceedings from the International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions,* Dagstuhl Castle, Germany, September 14, 1992.

Kemerer, Chris F., Rajiv D. Banker and Hsihui Chang. *Evidence on Economies of Scale in Software Development,* CISR WP No. 260, Sloan WP No. 3620-93, Center for Information Systems Research, Massachusetts Institute of Technology, July 1993.

Kemerer, Chris F., et al. "Software Complexity and Maintenance Costs," *Communications of the ACM*, Volume 36, Number 11, November 1993.

Keyes, Jessica. *Software Engineering Productivity Handbook,* Windcrest/McGraw-Hill, New York,

Knight, R.S. "Life Cycle Costing: An Industry View on the Way Ahead (Military Computing)," *Conference Proceedings MILCOMP 89,* Military Computers Systems and Software, p. 261-6, September 1989.

Lieberman, David. "Negotiating the Obstacles to Building Military Computers," *Computer Design,* Vol. 28, No. 11, p. 78-90, June 1989.

Lurie, Philip M. et al. *A Handbook of Cost Risk Analysis Methods,* Institute for Defense Analysis, April 1993.

McCord, J. W. *Software Development: A Product Life-Cycle Perspective*, Wright Research and Development Center, Wright-Patterson AFB, May 1990.

Meisner, R.E. *Software Product Factors for Development Cost Estimating at the Standard Systems Center,* Air Command and Staff Coll., Maxwell AFB, April 1988.

Messmer, Ellen. "Defense Pulls in Reigns on CALS Costs," *Network World*, September 9, 1991.

Moore, John. "New CALS Moniker Reflects Repositioning of Initiative (Continuous Acquisition and Life-Cycle Support Program)," *Federal Computer Week*, Vol. 7, No. 26, p. 19, September 26, 1993.

National Materials Advisory Board. *Enabling Technologies for Unified Life -Cycle Engineering of Structural Components,* Commission on Engineering and Technical Systems, March 1991.

Naval Weapons Engineering Support Activity, Management Engineering Department, Cost Management Division. *Life Cycle Cost Guide for Equipment Analysis*, November 1977.

Neely, E.S., R.D. Neathammer. "Building Life Cycle Costs in the United States Army," *Proceedings of the Third Conference on Human-Computer Interaction*, Vol. 1, p. 147-154, Sept. 1989.

Office of the Secretary of Defense, *Cost Analysis Improvement Group, Aircraft Operating and Support Cost Development Guide.* 15 April 1980.

Pappas, Kim. "Cost-Analysis Software is Aimed at Defense and Government Contractors," *PC Week*, Vol. 5, Issue 21, p. 24, May 24, 1988.

Riggs, Jeffrey L. and Denise Jones. "Flowgraph Representation of Life Cycle Cost Methodology - A New Perspective for Project Managers," *I.E.E.E. Transactions on Engineering Management,* Vol. 37, No. 2, May 1990.

Schaefer, Tom. "Air Force Cost Estimating Model. Interactive Software Model," *Proceedings of the Annual AAS Conference,* December 1991.

Stone, Harold S. "Life Cycle Cost Analysis of Instruction-Set Architecture Standardization for Military Computer-Based Systems," *IEEE Computer*, Vol. 12, No. 4, April 1979.

Wittner A.V. et al. *An Individual System/Organizational Cost Model*, Volume III, The Tactical Air Defense (Tad) Model : A Time Phased Isoc Application, Research Analysis Corp, January 1968.

Young (Arthur) and Co. *Impact of Low Cost Computing Technologies on the Department of Defense*, 18 April 1983.

# Software Assembly for Real-Time Applications Based on a Distributed Shared Memory Model*

David B. Stewart, Matthew W. Gertz, and Pradeep K. Khosla

Department of Electrical and Computer Engineering and
The Robotics Institute,
Carnegie Mellon University,
Pittsburgh, PA 15213

*Development time and cost of real-time applications can be significantly reduced by reusing software from previous applications. With today's systems, however, even if some software is reused, a large amount of new code is still required to create the "glue" which integrates modules created by programmers at different sites. A new software engineering paradigm, called "software assembly," is the process of developing an application simply by combining software modules from distributed libraries, without the need to write nor automatically generate any glue code. In this paper, the underlying framework to support software assembly of real-time applications is presented. The primary contribution is the notion of port-based objects, which combine object-based design with the port-automaton theory, in order to model and develop reconfigurable software modules. The integration of these modules in a distributed shared memory environment is possible through the use of a global state variable communication mechanism. Support for the framework has been implemented as part of the Chimera Real-Time Operating System. We have also designed a hypermedia user interface called Onika which allows real-time applications to be assembled graphically.*

## 1 Introduction

Transfer and reuse of real-time application software is difficult and often seemingly impossible due to the incompatibility between hardware and systems software at different sites. This has meant that new technology developed at one site must be reinvented at other sites, if in fact it can be incorporated at all. Technology transfer, therefore, has been a very expensive endeavor, and the reuse of software from previous applications has been virtually non-existent.

An an example, consider users who are developing a real-time application and may need new software algorithms developed at other sites. Currently, users may go to the library or search through the network for keywords, then find a book or journal article describing the mathematical theory or computer science algorithm, and perhaps also a description of the original implementation. After they have printed a copy of the paper, they read the article closely, then spend significant time writing, testing, and debugging code to implement the algorithm on their own system. Once that is done, they write more code to integrate the new algorithm into their application, and perform further testing and debugging. This process can easily take days or weeks of the person's time for each algorithm needed for their application, and thus take many months to complete the programming of an entire application.

The ultimate application programming environment, however, would allow for complete software reuse and the ability to quickly transfer technology from remote sites. There would exist a global distributed software library based on the information super-highway, similar to the hypermedia information system currently available on the World-Wide Web. Users who need the algorithm would search the library to find the appropriate book or article as is done today. However, when they find a suitable article, they not only get the theory from the article, but they can also follow a hyperlink to a reusable software module created by the authors, with the algorithm already programmed, fully tested, and debugged. With an action as simple as a mouse-click, that software algorithm is copied into the user's personal library, and is ready to be used in their application. This process would take a few minutes at most. The user could then assemble their software application by putting together these software building-blocks through the use of a graphical user interface. Within hours, a complete application could be assembled, as compared to the months that it would take to do so using conventional methods.

We propose the use of a software assembly paradigm to obtain the programming environment described above. Software assembly is the process of synthesizing an application without writing or automatically generating any new glue

The research presented in this paper is supported, in part, by Sandia National Laboratories, NASA, the Dept. of Electrical and Computer Engineering and the Robotics Institute at Carnegie Mellon University. Partial funding for David B. Stewart is provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) through a graduate fellowship. Partial funding for Matthew W. Gertz is provided by NASA Langley Research Center through a GSRP fellowship.

code. Software assembly is achieved by designing the underlying software as dynamically reconfigurable modules. A real-time operating system (RTOS) provides the services required to automatically integrate these modules in a shared memory distributed environment.

In Section 2, we first present related work in the area of software reuse. In Section 3 we introduce the notion of port-based objects, which combines object-based design with the port-automaton model, and which forms the basis of developing reconfigurable real-time software that can be assembled. In Section 4 we present the global state variable table mechanism, which we have designed to allow for the automatic integration of port-based objects in a distributed shared memory environment. Access to the global software database and assembly of the reconfigurable software modules is performed through a graphical user interface, as described in Section 5. Finally, in Section 6, we summarize our work on software assembly for quickly developing distributed shared memory real-time applications.

## 2 Related Work

There has been significant research in the area of software reuse, with three major directions emerging: software synthesis, interface adaptation, and object-based design.

*Software synthesis*, also known as automatic code generation, generally employs artificial intelligence techniques such as knowledge bases [1] [3] [18] and expert systems [11] [16] to generate the "glue" code for automatically integrating reusable modules. As input, they receive information about the software modules, the interface specifications and the target application, and as output produce code using both formal computation and heuristics. For a truly generic framework, however, it is desirable that the integration of software be based on the interfaces alone, and not on the semantics of the modules or application, as the latter results in the need for large knowledge bases. Furthermore, software synthesis only allows for statically configuring an application, and usually does not support dynamic reconfiguration.

*Interface adaptation* involves modifying the interfaces of software modules based on the other software modules with which they must communicate in order to obtain the required software integration. In these systems, an interface specification language is used to provide a general wrapper interface and to allow meaningful data to be interchanged between the modules [10] [12] [14]. This method has led to the notion of a software bus, where an underlying server or transport mechanism adapts to the software module, rather than having the software modules adapt to the transport mechanism [4] [15]. None of these methods have been adapted to real-time systems, and there are no clear extensions which would ensure that interface adaptation and communication between modules can be performed in real-time.

*Object-based design* addresses the root of the problem of software reuse by defining specifications for software module interfaces from the outset. An *object* is a software entity which encapsulates data and provides *methods* as the only access to that data [6]. Wegner distinguishes between two types of object design methodologies: *object-based design* (OBD) and *object-oriented design* (OOD) [25]. Whereas OBD only defines the encapsulation of data and access to that data, OOD is an extension which also defines the interrelation and interaction between objects. The interrelation of objects in OOD is defined through inheritance using the notions of classes, superclasses, and metaclasses [25]. An object-oriented programming language generally performs runtime dynamic binding to support this inheritance, and objects of different classes communicate with each other through messages, where the message invokes the method of another object. Such dynamic binding and message passing creates unpredictable execution delays, especially in a distributed environment, and as a result is not suitable for the design of real-time systems [5].

The Chaos Real-Time Operating System [17] was designed to use object-oriented design with real-time systems. Chaos addresses the the dynamic binding issue by performing static binding during the compilation and linking stages, thus allowing for predictable execution of the real-time application. Although object-oriented design is suitable for dynamically reconfigurable systems, the use of static binding for a real-time application eliminates that capability, and results in a system that is only statically configurable. The Chaos system addresses the real-time message passing issue by creating a variety of specialized messages which are tailored to the target application. As stated in [5] this, to some extent, ruins the object model's uniformity, and thus partially defeats the purpose of using the object-oriented methodology for developing real-time systems.

We have taken an alternate approach which avoids the real-time problems associated with object-oriented design, while maintaining the advantages of using objects for software reusability and reconfigurability. As described in the next section, we combine the advantages of object-based design with the port-automaton computational model to obtain the required real-time predictability.

## 3 Port-Based Objects

We combine the use of objects with the port-automaton formal computation model [24] for interaction between the objects, instead of classifying objects by their inheritance or by their interrelation through messages. A *port-automaton* is a model of a concurrent process, where an output response is computed as a function of an input response. The automaton executes asynchronously and independently, and whenever input is needed, the most recent data available is obtained. The automaton may have internal states; however all communication with other concurrent processes are through the ports. The port-automaton theory was first applied to robotics by Lyons and Arbib [13], who constructed a special model of computation based on it, which was called *Robot Schemas*. The schema used the port-automaton theory to formalize the key computational characteristics of robot programming into a single mathematical model. Arbib and Ehrig extended the work on robot schemas for algebraically specifying modular software for distributed systems by using *port specifications* to link modules [2]. The specification presented requires that there be exactly one input for every output link, and vice versa. The specification does not include any notions of objects in order to obtain reusability of the modules and reconfigurability of a task set, and there is no implementation presented that can map their specification into an actual system.

In our research, these port specifications have been combined with object-based design in order to create the *port-based object* model of a reconfigurable software module [19]. A port-based object is defined as an object, with various ports for real-time communication. As with any standard object [6], each module has a state and is characterized by its methods. The internals of the object are hidden from other objects. Only the ports of an object are visible to other objects. A simplified model of a port-based object is shown in Figure 1. Each module has zero or more *input ports*, zero or more *output ports,* and may have any number of *resource ports*. Input and output ports are used for communication between tasks in the same subsystem, while resource ports are used for communication external to the subsystem, such as with the physical environment, other subsystems, or a user interface.

A *link* between two objects is created by connecting an output port of one module to a corresponding input port of another module. We extend the port specifications as compared to that presented with the port-automaton theory so that an input port can be spanned into multiple outputs and outputs can be joined into a single input. In addition, operating system services are provided such that the communication through these ports can be performed in real-time and port-based objects can be reconfigured dynamically (see Section 4). A single output may be used as input by multiple tasks. In our diagrams, we represent such fanning of the output with just a dot at the intersection between two links, as shown in Figure 2. In that example, both modules $A$ and $B$ require the same input $p$, and therefore the module $C$ fans the single output $p$ into two identical outputs, one for each $A$ and $B$.

If two modules have the same output ports, then a join connector is required to ensure the integrity of the potentially conflicting data, as shown in Figure 3. A join connector is a special object which takes two or more conflicting inputs, and produces a single non-conflicting output based on some kind of combining operation, such as a weighted average. In this example modules $A$ and $B$ are both generating a common output $p$. In order for any other module to use $p$ as an input, it must only connect to a single output $p$. The user can modify the output ports of modules with conflicting out-

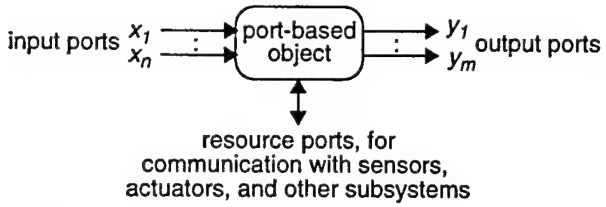



**Figure 1: Simplified model of a port-based object**

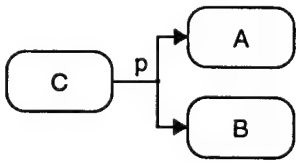



**Figure 2: Fanning an output into multiple inputs**

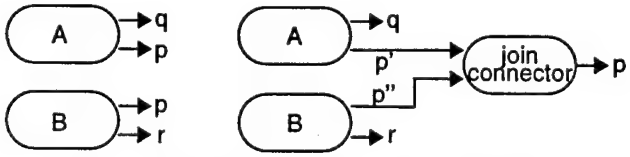



**Figure 3: Joining multiple outputs into a single input**

puts using the aliasing features provided by the RTOS and graphical user interface, such that they are two separate, intermediate variables. In our example, the output of module $A$ becomes $p'$, and the output of module $B$ becomes $p''$. The join connector takes $p'$ and $p''$ as inputs, and produces a single unambiguous output $p$.

A task is not required to have both input and output ports. Sensory-input modules, for example, only have output ports as their input is received from the environment through the resource ports. Similarly actuators generally only have input ports, and their output is transferred to the physical world through the resource ports. Other tasks may generate data internally or receive data from an external subsystem (e.g. trajectory generator and vision subsystem interface), or just gather data (e.g. data logger and graphical display interface) and hence not have input or output ports.

### 3.1 Configuration Verification

A task set is formed by selecting multiple objects which together form either an open-loop or closed-loop system. Each object executes as a separate task on one of the RTPUs in the real-time environment. An example of a fairly simple task set is the PID joint control of a robot, as shown in Figure 4. It uses three modules: the *joint position trajectory generator*, the *PID joint position controller*, and the *torque-mode robot interface*.

A legal configuration exists when there is exactly one output port for every input port in the task set, and that there are no two modules which produce the same output.

The correctness of a configuration can be verified analytically using set equations, where the elements of the sets are the state variables. A configuration is legal only if

$$(Y_i \cap Y_j) = \varnothing \text{, for all } i,j \text{ such that } 1 \le i,j \le k \wedge i \ne j \tag{1}$$

and

$$\left( \left( \bigcup_{j=1}^{k} X_j \right) \subseteq \left( \bigcup_{j=1}^{k} Y_j \right) \right) \tag{2}$$

where $X_j$ is a set representing the input variables of module $j$, $Y_j$ is a set representing the output variables of module $j$, and $k$ is the number of modules in the configuration.

### 3.2 Internal Structure

A port-based object contains several special methods which are called by the operating system in response to various signals. The methods allow for separate creation and activation of the software modules, periodic cycling or aperiodic handling of events, and error handling and recovery. Details of the internal structure of a port-based object, as well as C-language interface specifications and templates for developing software using the port-based object model, are given in [19]. Reconfigurable software modules created using the specifications can then be used in any application that makes use of the software assembly paradigm described in this paper.

## 4 Control Module Integration

In order to support our abstraction of port-based objects, a real-time mechanism which allows for the multi-threaded communication of multiple tasks in a multiprocessor system is required. In addition, the communication mechanism must be flexible for adding and deleting communication channels as the task set may be dynamically changing. The overhead of the mechanism must also be low, so as to not dominate usage of the CPU.

To address this problem, we have designed a *state variable table mechanism* (which we abbreviate as SVAR) for providing the real-time intertask communication of a task set in a distributed shared memory environment [21]. The com-

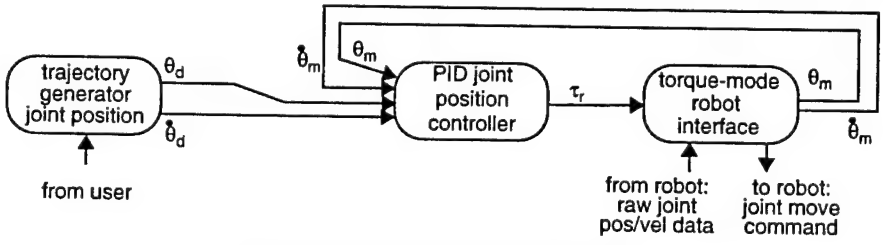



**Figure 4: Example of PID joint control.**

munication mechanism is based on the combined use of global shared memory and local memory for the exchange of data between modules, as shown in Figure 5. Every input port and output port is a state variable. A *global state variable table* is stored in the shared memory. The variables in this table are a union of the input port and output port variables of all the modules that can be configured into the system. Tasks corresponding to each control module cannot access this table directly. Instead, every task has its own local copy of the table, called the *local state variable table*. Only the variables used by the task are kept up-to-date in the local table. Since each task has its own copy of the local table, mutually exclusive access is not required. At the beginning of every cycle of a task, the variables which are input ports are transferred into the local table from the global table. At the end of the task's cycle, variables which are output ports are copied from the local table into the global table. This design ensures that data is always transferred as a complete set, since the global table is locked whenever data is transferred between global and local tables.

By using global state variables, tasks can be developed independent of the target application, as the only requirement is that the input constants and variables required by that task are produced by some other task. The underlying operating system mechanisms take care of all of the synchronization and setting up the communication paths to ensure that the data is at the correct place when it is required. Furthermore, tasks can be dynamically reconfigured, as a new task being swapped into the system immediately has access to the constants and variables which were used by the task being swapped out of the system.

The global state variable table mechanism and other support for integrating port-based objects have been implemented in the Chimera RTOS [20] [23]; analysis of the mechanism and performance benchmarks are given in [19].

## 5 Software Assembly through Graphical User Interfaces

Software assembly of reconfigurable software modules is performed by the user through a hypermedia user interface which we call Onika [7]. In this section, we give a brief overview of Onika. The reader who wishes more in-depth information should refer to [9].

Onika is a visual programming environment developed at Carnegie Mellon University. It allows the user to directly control and manipulate tasks as implemented in the Chimera RTOS. The port-based objects are stored in software libraries which may be either remote or local. When Onika is launched, it searches user preferences for hyperlink anchors to software libraries. Modules from remote libraries are downloaded and automatically linked with the local modules into a Chimera executable as needed. An iconic hyperlink is generated on-the-fly for each task, and is displayed to the user within a library window. The user can retrieve a variety of information about any given task by clicking on its icon, and can also modify the modules through this hyperlink (if allowable). The icon is displayed as a port-based object, showing its input and output ports clearly, as well as its current status, its name, and its frequency.

Onika links with Chimera via a network, using two sockets. One socket is for bi-directional synchronous communication, and is used to issue commands and receive acknowledgments and data from Chimera. The other socket is unidirectional and used by Chimera to send signals to Onika, including error signals and special user-defined signals such as "job completed".

The user creates configurations (also called *jobs* in Onika) by placing icons from the library onto a job canvas. Onika sends a command to Chimera to spawn a new task for each icon placed onto the canvas. The tasks automatically inter-

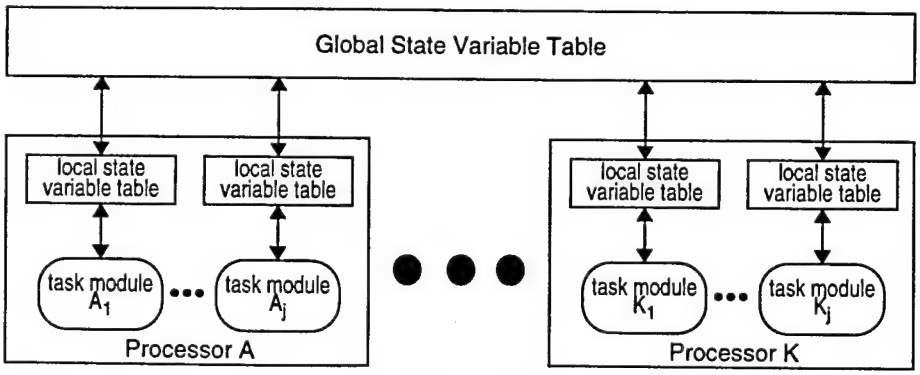



**Figure 5: Structure of state variable table mechanism for control module integration**

connect graphically with those which share common data ports already placed into the configuration, and represent a direct mapping to the global state variable table communication channels that exist in the real-time environment. The configurations are thus assembled graphically, and can be saved for later recall. The individual tasks can be turned on and off with simple mouse-clicks, and killed by selecting and deleting their icons. Certain task parameters can be changed easily, such as frequency, configuration constants, and aliases applied to I/O ports which do not match the local naming conventions. A status window gives more detailed information on the current status of real-time tasks and values of state variables.

Assuming that configurations have been previously saved, reconfiguration into a new job from the current job can be achieved within Onika with a simple mouse-click. The user specifies the configuration which should be loaded from a file navigator. The saved configuration contains hyperlinks to the various modules it needs. Onika reads these in, and compares the links of the two configurations. It then constructs the sequence of events necessary to dynamically reconfigure from the current job to the next. These instructions are sent to Chimera in the form of command packets, and the real-time operating system reconfigures to the new job. Onika checks Chimera's reply to determine if the configuration was successful, and updates the job canvas and the status information to reflect the current configuration.

Figure 6 shows an example of an Onika job canvas. Each icon represents a port-based object. Shaded icons have been created and activated, while white icons are created but not activated.

The programmer can create a pictorial iconic hyperlink to a configuration, which can then be stored in a higher-level library called a *job dictionary*. If the job requires some user input for execution (such as the desired endpoint in a joint motion job), this information can be pre-saved in a user I/O object, which is also assigned a pictorial icon and stored in the dictionary. Both jobs and objects can be viewed or edited by clicking on these pictorial anchors. Syntax and semantics are made apparent by the color and shape of each icon's edges. Jobs and objects are arranged sequentially, fitting together like puzzle pieces, in order to form an application, as shown in Figure 7. When executed, Onika uses dynamic reconfiguration to traverse the application. A job which is finished (e.g. the trajectory endpoint has been reached) sends a signal to Onika, which then proceeds to the next job in the sequence.

Onika and Chimera have been used several times to transfer technology and control multisensor systems during demonstrations of a virtual laboratory to top-level administrators and scientists at Sandia National Laboratories. Details of using software assembly for virtual laboratories can be found in [8].

## 6 Summary

In this paper we introduce the notion of software assembly for real-time applications which are implemented in a distributed shared memory environment. The primary contribution is the notion of port-based objects, which combine object-based design with the port-automaton theory, in order to model and develop reconfigurable software modules. The real-time communication between port-based objects is performed through a global state variable mechanism that has been incorporated into the Chimera Real-Time Operating System. Software assembly and control of multi-sensor systems is performed by the user through a graphical user interface which we call Onika. Onika has been used to transfer technology in a virtual laboratory environment.

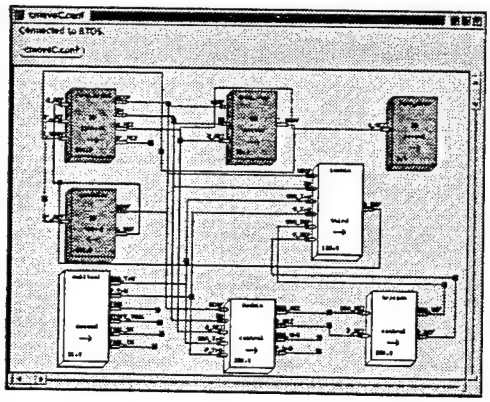


**Figure 6: An example of a configuration of tasks assembled within Onika's job canvas**

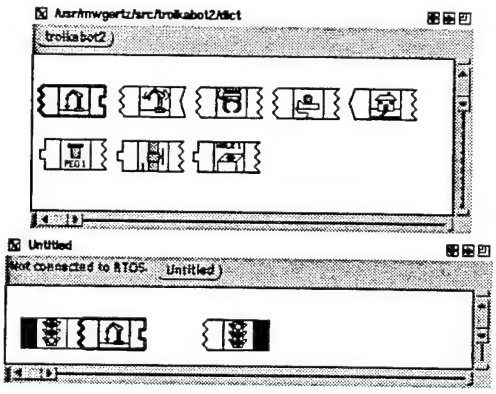



**Figure 7: An example of an Onika job dictionary (above) and application workspace.**

## 7 References


[1] B Abbott et al, "Model-based software synthesis," *IEEE Software*, pp. 42-52, May 1993.

[2] M. A. Arbib and H. Ehrig, "Linking Schemas and Module Specifications for Distributed Systems," in *Proc. of 2nd IEEE Workshop on Future Trends of Distributed Computing Systems*, Cairo, Egypt, September 1990.

[3] D. Barstow, "An experiment in knowledge-based automatic programming," *Artificial Intelligence*, vol.12, pp. 73-119, 1979.

[4] B. W. Beach, "Connecting software components with declarative glue," in *Proc. of International Conference on Software Engineering*, Melbourne, Australia, pp. 11-15, May 1992.

[5] T. E. Bihari, P. Gopinath, "Object-oriented real-time systems: concepts and examples," *IEEE Computer*, vol. 25, no. 12, pp. 25-32, December 1992.

[6] G. Booch, "Object-oriented development," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 2, pp. 211-221, February 1986.

[7] M.W. Gertz, D. B. Stewart, and P. K. Khosla, "A Software architecture-based human-machine interface for reconfigurable sensor-based control systems," in *Proc. of 8th IEEE International Symposium on Intelligent Control*, Chicago, Illinois, pp. 75-80, August 1993.

[8] M.W. Gertz, D.B. Stewart, B. Nelson, and P.K. Khosla, "Using hypermedia and reconfigurable software assembly to support virtual laboratories and factories," in *Proc. of 5th International Symposium on Robotics and Manufacturing (ISRAM)*, Maui, Hawaii, August 1994.

[9] M. W. Gertz and P. K. Khosla, "The Onika User's Manual," Program Documentation, Dept. of Elec. and Comp. Engineering and The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 (e-mail *chimera@cmu.edu* for a copy).

[10] N. Haberman, D. Notkin, "Gandalf: software development environments," *IEEE Transactions on Software Engineering*, vol.12, no.12, pp. 1117-1127, December 1986.

[11] R.K. Jullig, "Applying formal software synthesis," *IEEE Software*, vol.10, no.3, pp. 11-22, May 1993.

[12] D. A. Lamb, "IDL: Sharing intermediate representations," ACM Transactions on Programming Languages and Systems, vol.9, no.3, pp. 297-318, July 1987.

[13] D. M. Lyons and M. A. Arbib, "A formal model of computation for sensory-based robotics," *IEEE Transactions on Robotics and Automation*, vol 5, no. 3, pp. 280-293, June 1989.

[14] J. Magee, J. Kramer, M. Sloman, and N. Dulay, "An overview of the REX software architecture," in *Proc. of Second IEEE Workshop on Future Trends of Distributed Computing Systems*, Cairo, Egypt, pp. 396-402, September 1990.

[15] J. M. Purtilo and J. M. Atlee, "Module reuse by interface adaptation," *Software–Practice and Experience*, vol.21, no.6, pp. 539-556, June 1991.

[16] C. Rattray, J. McInnes, A. Reeves, and M. Thomas, "Knowledge-based software production: from specification to program," in *UT IK 88 Conference Publication*, pp. 99-102, July 1988.

[17] K. Schwan, P. Gopinath, and W. Bo, "Chaos: kernel support for objects in the real-time domain," *IEEE Transactions on Computers*, vol. C-36, no. 8, pp. 904-916, August 1987.

[18] T. E. Smith and D. E. Setliff, "Towards an automatic synthesis system for real-time software," in *Proc. of Real-Time Systems Symposium*, San Antonio, Texas, pp. 34-42, Dec. 1991.

[19] D. B. Stewart, *Real-Time Software Design and Analysis of Reconfigurable Multi-Sensor Based Systems*, Ph.D. Dissertation, Carnegie Mellon University (Pittsburgh, PA) April 1994.

[20] D. B. Stewart, D. E. Schmitz, and P. K. Khosla, "The Chimera II real-time operating system for advanced sensor-based robotic applications," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1282-1295, Nov./Dec. 1992.

[21] D. B. Stewart, R. A. Volpe, and P. K. Khosla, "Integration of real-time software modules for reconfigurable sensor-based control systems," in *Proc. 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '92)*, Raleigh, North Carolina, pp. 325-333, July 1992.

[22] D. B. Stewart, R.A. Volpe, and P.K. Khosla, "Design of Dynamically Reconfigurable Real-Time Software using Port-Based Objects," Technical Report CMU-RI-TR-93-11, Dept. of Elec. and Comp. Engineering and The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213.

[23] D. B. Stewart and P. K. Khosla, *Chimera 3.1 Real-Time Programming Environment*, Program Documentation, Dept. of Elec. and Comp. Engineering and The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213; to obtain copy electronically, send email to <*chimera@cmu.edu*>; July 1993.

[24] M. Steenstrup, M. A. Arbib, and E. G. Manes, "Port automata and the algebra of concurrent processes," *Journal of Computer and System Sciences*, vol. 27, no. 1, pp. 29-50, August 1983.

[25] P. Wegner, "Concepts and paradigms of object-oriented programming," *OOPS Messenger*, vol.1, no.1, pp.7-87, August 1990.

# Synthesis of Future Submarine Command and Control System Architectures

Steve Harrison Naval Undersea Warfare Center Newport Division
email: harrison@code20.nl.nuwc.navy.mil Phone: (203)440-6153

## Abstract

The Navy needs to reduce the cost of the development and maintenance of command and control systems. Cost is driven by many factors including the use of special purpose, unique military hardware which is expensive to maintain as well as procure. Additionally, the replacement of the hardware as it becomes obsolete usually requires the costly development of new software for each new generation of hardware.

In order to reduce costs, command and control systems are being implemented with more and more Open System Architecture (OSA) and Commercial Off-The-Shelf (COTS) components. This paper describes an approach to cost effectively mitigate risks associated with the use of OSA and COTS products in command and control systems. The approach is to demonstrate tactical capabilities that have been implemented using OSA/COTS components in a series of industry/government demonstrations. The demonstrations are experiments that test approaches proposed by industry and government. The experience gained through this process can be used to specify, evaluate, and test OSA components and to reduce the risk associated with the introduction of OSA and COTS components in command and control systems.

The NSSN (New, Ship, Submarine, Nuclear) program is evaluating the application of OSA and COTS technology to submarine command and control systems using the approach described above. The results of the first demonstration and the plans for the second demonstration are summarized.

## Introduction

Previously, new command and control systems were developed using custom components specified and built to stringent military standards to meet shock, vibration, noise and environmental requirements as well as increasing performance requirements. The Navy can no longer afford to develop special purpose hardware and software as the first choice.

MIL-STD-2036 mandates the use of **Non-Development Items** (NDI) (Bold items are defined in a glossary at the end of this paper) over the development of new components. NDI includes previously developed **militarized** hardware, **ruggedized Commercial Off-The-Shelf** (COTS), **modified COTS** and **COTS** components that provide the appropriate reliability and **operational availability** required for command and control system missions. Program managers decide whether NDI is capable of meeting system requirements. This change in philosophy opens the options available to developers and program managers.

The submarine community is developing approaches to take advantage of these new options[1]. It is certain that the command and control system will have to evolve. The decisions of how to evolve the system are complex and careful analysis is required. A long term end goal is needed to steer these decisions toward a cost effective, maintainable system. The end goal system is envisioned to be based on a mixture of military hardware, ruggedized COTS and COTS equipment that conforms to **Open System Architecture** (OSA) interface standards so that the hardware is easily replaceable over the system's life

cycle. The most important characteristics of the end goal architecture are the portability of tactical applications and the interoperability of various subsystems.

The NSSN program is investigating options for introducing OSA based components into the NSSN command and control system. Industry initiatives are driving the efforts to investigate the suitability of OSA /COTS based systems to meet tactical requirements. The issues to be addressed include real time performance, interoperability, software portability, reuse of tactical software, compatibility of OSA interfaces to existing military systems, ability to meet tactical processing and communication requirements and reliability and availability characteristics.

In order to investigate the problems associated with the eventual evolution of the command and control system, available military components are used as a demonstration baseline. The demonstration baseline is evolved to meet new requirements and replace obsolete equipment by **reuse, reengineering** or **reimplementation**. The cost, level of effort, performance and other metrics are observed in order to understand new problems and avoid them in the actual NSSN development.

## Open System Architectures and Submarine Systems

Open Systems Architectures are an important part of the new DOD acquisition strategy. The approach proposed is based on the incorporation of NDI and COTS to provide an Open System Architecture[2]. The migration path toward an Open System must be based on cost effective strategies that provide the required mission capabilities during the transition.

In the past the maintenance of hardware has been stressed because it was the most expensive component. Software was custom built to fit on the scarce hardware resources. As hardware has decreased in cost and software has greatly increased in size, cost and complexity, the situation has reversed. To save money, the software applications must be reused while the hardware is replaced periodically. Software that complies to Open System Standard interfaces retains value because it will be compatible with future hardware technologies that may be used.

The new approaches required for synthesizing submarine command and control system architectures closely match the objectives of the Next Generation Computer Resources (NGCR) program. The NGCR program objectives include increased Weapons Systems operational readiness and interoperability, rapid adaptation to changing threats, competitive product development and system upgrades, incorporation of affordable and current technology, and increased program management flexibility[3]. This program includes plans to establish interface standards for backplanes (FUTUREBUS+), Communications networks (SAFENET), Operating Systems (POSIX), Graphics Language/Interfaces, Database Management Systems and High Speed Data Transfer Network (HSDTN). Some of these interface standards are in place and others will be available in the future. The NGCR program builds on current trends and standards used in the commercial market. The Open System Architecture approach is being employed to an increasing level in the commercial market. The concept is to employ non-proprietary standards to allow system integration based on multi-vendor components. These components must comply with the same standards for interfaces. Naval Undersea Warfare Center (NUWC) is incorporating these standards in policies and guidance for using OSA components within the NSSN program as part of the NSSN Functional Requirements Document.

## OSA Demonstration Objectives

The objectives of OSA demonstrations are to learn how to specify, evaluate and test tactical implementations based on OSA and COTS components and to reduce the risk of introducing OSA components into command and control systems. It is also very important to demonstrate the cost effectiveness of evolution of the current custom built military components to OSA and COTS based implementations.

Cost effectiveness is dependent upon the ability to save money by retaining application software while replacing obsolete hardware. Portability of applications is the inexpensive re-hosting of software from one computer (presumably obsolete) to another (presumably more powerful and less expensive). Interoperability is the cost effective integration of software and hardware components to work as a system. These capabilities are essential to realize savings.

In the past, computer performance issues have caused cost and schedule overruns. Computer performance models are capable of estimating the computer performance of processing elements, networks and systems[4]. An objective of OSA demonstrations is to understand and develop capabilities to predict computer performance of OSA components in order to evaluate design options, identify risks, and develop test criteria.

## Evolution through Reuse, Reengineering and Replacement

As previously stated cost constraints are driving designs for new systems toward a synthesis approach based on NDI. The approaches to synthesis are the adaptation of existing NDI components as a system baseline and the evolution of that system toward a high performance low maintenance Open System. The evolution process takes three approaches: reuse, reengineering and reimplementation.

## Open System Architecture Demonstrations

The proponents of OSA technologies have promised a means of cost effectively evolving systems toward modern, powerful and less expensive implementations over the system's life cycle. The OSA promises include: a) portability of application software, b) interoperability of components from various vendors that conform to the same interface standards, c) upgrade paths based on future high performance components, and d) low cost components that will be available in the market place.

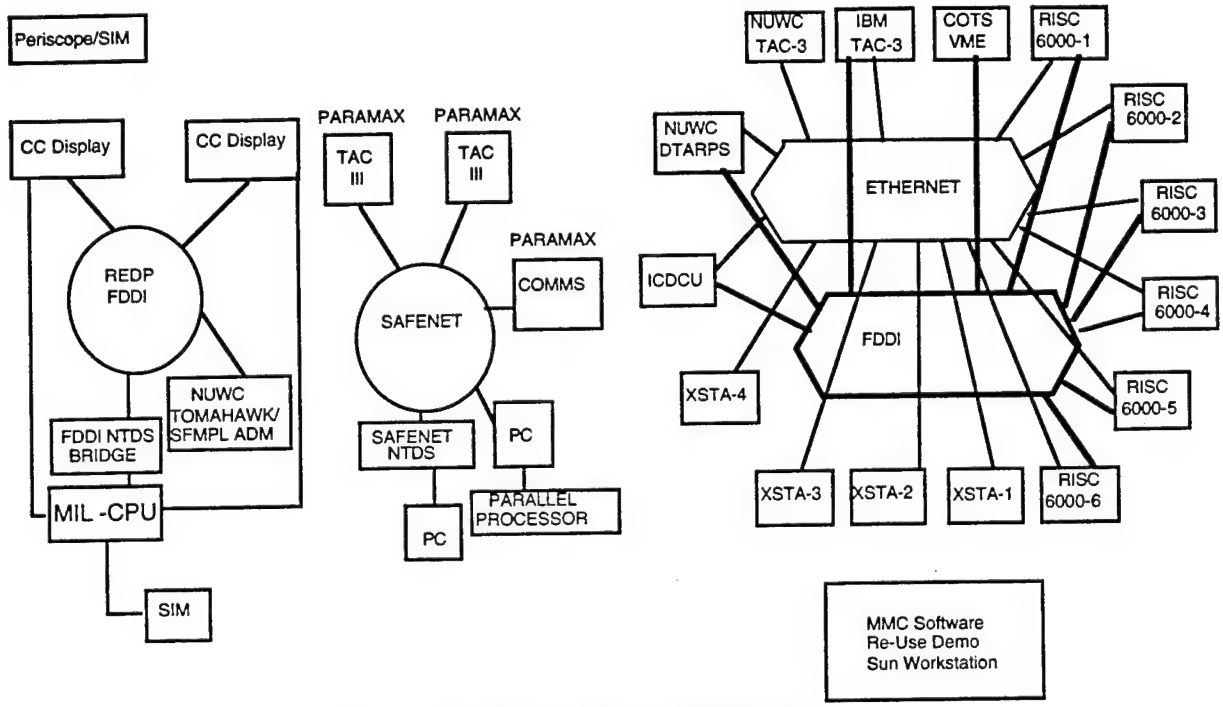
The Navy is attempting to evaluate these promises and determine how to specify systems, evaluate designs, and test implementations based on OSA components through the OSA Demonstration process for the NSSN program. During Phase I, several industry and Navy participants supported these efforts by providing technical support and providing a demonstration item.

## NSSN OSA Demonstration

The NSSN OSA demonstration was proposed during early FY93. The major submarine command and control system vendors were briefed and solicited for proposals. Several small multi-year Broad Agency Announcement (BAA) contracts were awarded to submarine command and control system industry vendors. NUWC provides coordination, testing, and reporting.

## Phase I Summary of Results

The Phase I demonstration[5] was successfully conducted in September of 1993 at NUWC Division Newport, RI. An overview of the Phase I demonstration is provided here. Figure (1) shows the Phase I OSA configuration.




Figure(1) Phase I OSA Configuration

IBM Federal System Division (FSD) demonstrated reimplementation of Towed Array processing using OSA and COTS components. The software was ported between various platforms and the level of effort required was quantified. Performance measures of processing and communications were made. In general, the OSA component met tactical display rates and portability seemed to be cost effective.

NUWC provided a demonstration of a OSA and COTS based towed array processing. The NUWC demonstration used sensor data generated by the IBM FSD demonstration through an FDDI interface. NUWC also provided support for performance measurements during Phase I. These measurements were used to construct a model of the IBM FSD demonstration. This model will be used to predict performance of the Phase II demonstration.

Martin Marietta Corporation (MMC) demonstrated the reuse of tactical Ada code ported from tactical processors to an OSA platform. A tactical database application with support software was ported using a software shell to convert tactical operating system calls to POSIX compliant calls. A previously developed critical item test was executed on the OSA/COTS platform. The functionality of the software was demonstrated and compared to tactical test results. The ported software provided the same functionality but the performance was below expectations due to the lack of a multi-threaded operating system. The operating system used in the MMC demonstration did not allow for the

concurrent execution of Ada tasks. MMC plans to use a multi-threaded operating system in Phase II.

Unisys demonstrated the initialization, reconfiguration, connection-less, connection oriented and multi-cast capabilities of a small SAFENET LAN. The synchronization of time and the communication performance and latency was also measured. Unisys also provided an interoperability demonstration of a command control center using military radio receivers connected to OSA components.

Raytheon Equipment Division Portsmouth, RI (REDP) provided a demonstration of reengineering by distribution of a portion of the combat control database function from a military CPU to OSA based workstations using an FDDI LAN.

## Phase II Plans

The plan for Phase II emphasizes interoperability between subsystems. The industry participants are continuing and expanding the Phase I investigations. The new areas of the plan are summarized below. Phase II expands on the Phase I configuration as shown in Figure (2).

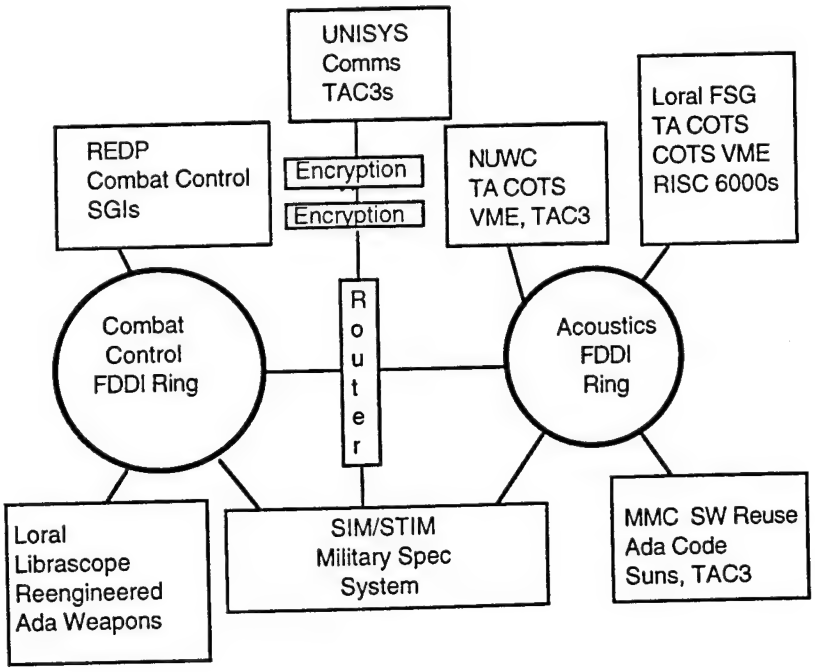



Figure (2) Simplified Phase II OSA Demonstration Configuration

Loral FSG (formerly IBM FSD) is planning to concurrently display tactical Spherical Array (SA) data (on a two display surface reengineered display) with other OSA acoustics displays. MMC is extending their demonstration to provide Wide Aperture Array processing based on reused Ada code ported to SUNs and TAC-3 workstations. Loral FSG, NUWC and MMC will provide tracker data to REDP combat control across an FDDI LAN.

Unisys and NUWC will provide a demonstration of communication with external platforms for receiving targeting data and exchanging tactical communications. Navigation and real time distribution will be investigated with NUWC support.

REDP combat control demonstration will provide interoperation with the acoustics demonstrations by accepting tracker data. REDP will also except targeting data from the communications demonstration. Tracker data and communications data will be used to develop targeting solutions that will be sent to weapons demonstrations.

Loral Librascope is reengineering tactical weapons Ada code by replacing tactical operating system calls with OSA POSIX and X standard interfaces. The modified weapons software will accept targeting data from combat control and simulate weapon firing.

## Issues

The issues identified through the OSA process from the perspective of the Technical Direction Agent role are focused on how to specify, evaluate and test OSA components.

Although COTS/OSA components offer large performance gains over existing tactical computers, there are some real time tactical requirements that must be carefully evaluated. The tactical requirements of concern include the ability to quickly start up or reconfigure the system in a specific operational state. In addition, tactical applications require real time operating systems services and deterministic behavior. There are new versions and extensions of the existing OSA standards that address these issues. They will be investigated as part of the OSA demonstration process.

The Man Machine Interfaces (MMI) of current tactical workstations are different from available OSA and COTS components. For example, some tactical workstations use four trackmarbles, dedicated displays for fixed function keys, special purpose command action entry panels and finger on glass input devices. The issue is how to map these MMIs to OSA standard Graphical User Interfaces in such a way that they are portable to the next generation of OSA workstations.

## Conclusion

The OSA demonstration approach has provided a testbed for studying the issues associated with specifying, evaluating and testing OSA components in submarine command and control systems. This approach allows the mitigation of risk associated with the introduction of OSA/COTS components. The effort is an important part of the system engineering process for the synthesis of large complex systems from NDI components. This process is continuing and will allow the Navy to better specify, evaluate and test submarine command and control systems synthesized from existing tactical systems and OSA/COTS components.

## ACKNOWLEDGMENTS

Support for this paper has been provided through the Engineering Complex Systems Block Program manager Steve Howell and technical lead Mike Jenkins of NSWC. PEO SUBX NSSN Program Office has sponsored the NSSN OSA Demonstration. Rother Hodges, Robert Watson, Paul Gracia, Greg Majewski and other NUWC personnel have provided technical guidance concerning Open Systems Architecture and Next Generation Computer Resources as they apply to Submarine Combat Systems. Blake Ross of Loral FSG and Susan Bennett and Dave Levan of Martin Marietta Corporation as well as other industry participants have provided technical data and insights.

## REFERENCES

1. S.J. Harrison, "Evolution and Trends in Submarine Sonar System Architectures", U.S. Navy Journal of Underwater Acoustics, Volume 43, No. 4. October 1993.

2. P. A.Strassman., "Realizing the Economic Value of Open Systems", Proceedings of the Federal Open Systems Conference - 1991, p.13.

3. H. Mendenhall, " "Next Generation Computer Resources Program Description, "Proceedings of the Meeting The Challenges of the Future Conf. (4-6 Feb 1992).

4. S.J. Harrison, "Development of Sonar System Architectures using Analysis with Modeling, Benchmarking, Prototyping, and Critical Item Testing", U.S. Navy Journal of Underwater Acoustics, Volume 43, No. 4. October 1993.

5. P. Gracia, et. al., "Assessment Report for New Attack Submarine Open System Architecture Phase I Demonstrations," NUWC Technical Memorandum 30 Sept. 1993.

**DEFINITIONS** (From Mil-STD-2136 unless otherwise specified)

COTS: Commercial off the shelf. Items or equipment which can be purchased through commercial retail or wholesale distributors as is (for example, equipment that is available as a cataloged item).

Militarized: Those items which are specified and manufactured to military specifications and shall withstand all environmental conditions which may be encountered during wartime service.

Modified COTS: COTS equipment that has been customized to meet functional requirements. COTS equipment performance includes modified COTS items unless otherwise indicated. Ruggedized is COTS or modified COTS equipment that is modified to meet specified service requirements.

NDI: Non developmental item. NDI equipment can be COTS, ruggedized or militarized. NDI shall be defined as any of the following:

(a) Item of supply that is available in the commercial marketplace.

(b) Previously developed item of supply that is in use by a department of agency of the United States, a state or local Government, or a foreign Government with which the United States has a mutual defense cooperation agreements.

(c) Item described above that requires only minor modification to meet the procuring agency's requirements.

(d) Item currently being produced that does not meet the above requirements solely because it is not yet in use, or not yet available in the commercial marketplace (Section 907 of the Defense Acquisition Improvement Act of 1986).

Operational Availability: The expected percentage of time that a weapon system or individual equipment will be ready to perform satisfactorily in an operating environment when called for at any random point in time.

OSA: Open systems architecture. OSA shall be defined as design approach whereby hardware/software are designed to non-proprietary standards to allow the interfacing of components and systems manufactured by multiple vendors.

Reengineering: The examination and alteration of an existing subject system to reconstitute it in a new form. The process encompasses a combination of such as reverse engineering, restructuring, re-documentation, forward engineering and re-targeting. (Draft Mil-Handbook on Reengineering)

Reimplementation: The removal of an obsolete or inadequate component and replacement with new components. New components should be based on OSA and COTS technologies if they meet mission requirements. As a last resort custom built military hardware is used.(author's definition) .

Reuse: The incorporation of existing specifications, documentation, designs, code, and tests for software and hardware components. (author definition)

Ruggedized: COTS or modified COTS equipment that is modified to meet specified service requirements. Modified COTS involves modifications to meet functional requirements; Ruggedized incorporates modifications to meet service requirements. This may be in the form of added parts, such as shields and shock mounts, power conditioners, and so forth, or in the form of direct modification to COTS equipment.

# A Visual Platform for the Synthesis of Complex Systems

Michael Gorlick* Alex Quilici†

## Abstract

Visual programming research has largely focused on the issues of visual programming-in-the-small. However, entirely different concerns arise when one is programming-in-the-large. We present a visual software engineering environment, based on weaves, that allows users to construct visually programs consisting of hierarchically organized networks of components that process streams of arbitrary objects. We discuss the problems that occur when trying to construct complex systems consisting of thousands of interconnected components, examine how this environment deals with some of the problems specific to visual programming-in-the-large, and show why our initial solutions failed to scale successfully. Finally, we argue that a single visual mechanism called "zooming" addresses these scaling problems and, when suitably augmented, can also be used to support automatic component discovery and some forms of error correction.

## 1 Introduction

Software engineering as a discipline arose from the realization that programming-in-the-large is fundamentally different from programming-in-the-small [3]. In particular, the migration from small programs (those composed by a single person over a short period of time) to large systems (those composed by teams over a period of several years) had led to research into software development environments that address the following issues:

- constructing systems by composing a large number of multilingual modules and interconnections;
- specifying module interconnections in a manner amenable to automated checking, transformation, and verification;
- locating and reusing components at multiple levels of abstraction;
- providing ready access to ancillary information such as requirements, documentation, briefings, and engineering notes;
- browsing the system at multiple levels of abstraction simultaneously;
- refining specifications to implementations;
- modifying the system to support instrumentation, testing, and analysis; and
- locating and repairing errors in module interconnections.

For example, Arcadia [10] emphasizes component integration, and language processing, testing and analysis, and Inscape [16] focuses on the constructive use of formal module specifications to deal with the problems of programming-in-the-large.

Unfortunately, visual programming environments have by and large ignored these issues. Much visual-programming research has focused on providing support for visual programming-in-the-small, such as efforts to generate new visual programming languages [12], or on supporting specific software engineering tasks, such as software maintenance, by extracting and visualizing structural code information [2, 18]. While workers have started to consider the problems inherent in visual programming-in-the-large, they have focused on the specific issue of structured design [13] or hierarchical program development [11]. Not surprisingly, some of the issues of visual programming-in-the-large are receiving attention from the vendors of commercial visual programming languages such as Prograph and LabView.

This paper describes a visual programming environment, for weaves, that we are extending toward a *visual software engineering environment* by explicitly providing mechanisms to address the issues of in "visual programming-in-the-large." Section 2 describes weaves, their current development environment, and our first approach to two nagging software engineering problems: how to access supporting information,

*The Aerospace Corporation, P.O. Box 92957, Los Angeles, California 90009, (310) 336-8661, gorlick@aero.org

†University of Hawaii at Manoa, Department of Electrical Engineering, 2540 Dole St, Holmes 483, Honolulu, HI 96822, (808) 956-9735, alex@wiliki.eng.hawaii.edu

and how to browse large systems. Section 3 introduces the concept of zooming and describes a fresh approach to these problems. Sections 4 and 5 outline extensions to zooming that permit a visual solution to two other classic software engineering problems: component retrieval and the interactive correction of user errors. Finally, Section 7 distills our experience into a set of principles to guide developers of visual software engineering environments.

## 2 Weaves

We have implemented a visual software composition and integration environment for constructing systems as weaves. Weaves are networks of components in which streams of arbitrary objects flow from one component to another. They occupy a computational niche midway between fine-grain dataflow and large-grain stream processing (as exemplified by Unix pipes and filters). The weave visual editor, *Jacquard*, provides users with mechanisms for rapidly assembling weaves from components, executing and observing weaves, and combining and modifying weaves dynamically — all using nothing but point, click, drag, and drop.

Figure 1 shows a portion of a stereo tracker implemented as a weave. Weaves are comprised of sockets (which are either unpopulated or populated), components, and jumpers. Unpopulated sockets are placeholders for components that consume objects as inputs and produce objects as outputs. Jumpers between sockets provide transport services for moving objects from one place to another and thus define the topology of the network. Users assemble weaves by interconnecting sockets and populating each socket with a component (which may itself be a weave). Type information about the objects flowing through a connection is specified by labeling the jumper. Figure 2 shows a small weave being built with Jacquard. Weaves are well suited for systems characterized by processing on continuous or intermittent streams of data, and they have been applied to such tasks as satellite telemetry processing, tracking, and the rapid prototyping of satellite ground stations. A detailed discussion of the computational and communication semantics of weaves can be found in [8].

Weaves were specifically designed to tackle the problem of constructing large systems by composing components and interconnections, the visual equivalent, if you will, of a module-interconnection language. However, particular attention has been paid

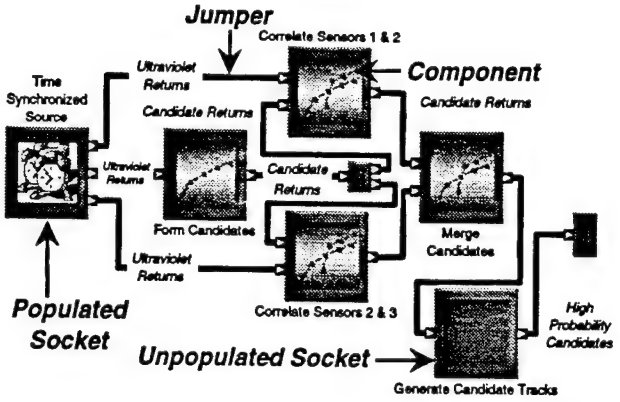



Figure 1: A portion of a weave-based stereo tracker.

to other outstanding and troublesome software engineering problems such as access to large quantities of diverse supporting information and structured browsing through multiple levels of a system. We briefly describe those aspects here, indicate how they fail to scale, and then, in Section 3, propose a single visual mechanism, *zooming*, to remedy the difficulties.

### 2.1 Weave Services

Weaves have a variety of unique features that make them especially attractive for constructing complex systems that must be implemented on distributed or heterogeneous architectures:

- Weaves allow streams of arbitrary objects to be sent from one weave component to another. This allows legacy software to be wrapped and included as weave components. In contrast, other systems such as AVS [19] limit the data exchange to a small set of primitive data types, such as integers or reals or arrays of these types.
- Weaves can be edited and controlled while running, including dynamic snipping, splicing, and suspending of weave components. In contrast, other systems such as ConMan [9] only allow editing while the network is being constructed and do not allow dynamic control of execution.
- Weaves need not be fully specified to execute, although they then execute with reduced functionality. In contrast, other systems require users to completely specify all of the network components before the network can execute.
- Weaves allow low-overhead monitoring and observation by providing inexpensive built-in instru-

mentation and by allowing users to transparently introduce more specialized instrumentation into an executing weave. In contrast, other systems do not provide default or dynamic instrumentation.

- Weaves support parallelism and fault tolerance by allowing users to dynamically rearrange their execution structure (ie, to compensate for hardware failures or to take advantage of additional computing resources). In contrast, other systems have a static execution structure.

- Weaves support real-time processing by allowing their components to be distributed, by using lightweight processes and highly-efficient communication protocols, and by allowing arbitrary length streams of objects. In contrast, other systems such as Khoros [5] use heavyweight processes, often use inefficient communication protocols, often require that all data fit entirely in memory, and offer limited overlap between communication and computation.

- Weaves allow components to be composed in a variety of sequential programming languages including Fortran, C, C++, and Objective-C (encapsulated by suitable "glue" code) while jumpers provide a wide variety of interconnection semantics. Since weaves are indifferent to the composition of the components and seamlessly support forms of interconnection that can be tailored to an application they are well suited for the piecewise composition of large systems.

- Weaves encourage the hierarchical composition of systems by allowing weaves to themselves be encapsulated and used as components in higher-level weaves. The internal structure of such components is accessible in the same visual manner as the higher-level structures and all of the direct manipulations described above can be applied to an encapsulated weave.

All components in a weave adhere to the three principles of *blind communication*:

- *No component is aware of the source(s) of its input objects or the destination(s) of its output objects.* Consequently no component can determine its location in the topology of the network and may be freely moved about modulo its input and output expectations.

- *No component is aware of either the semantics of the transport service(s) used to deliver its input object or the semantics of the transport service(s) used to transmit its output objects.* Consequently all components are transport service independent and new transport services may be substituted for old without concern for the functional behavior of the components to which it is connected.

- *No component is ever aware of the loss of a connection.* Consequently weaves may be edited and modified during execution. Components can be excised or added and connections may be rerouted without fear that an object will be lost in mid-transmission and to the extent that the network of components can continue to execute it will. Weaves enjoy the same degree of plasticity at runtime that they do during development thereby granting application developers an unusual degree of flexibility.

## 2.2 Access to Supporting Information

One ongoing problem in the development of large military and commercial systems is the tidal wave of documentation that is required to specify, annotate, explain, and demonstrate their rationale, design, and structure. The current weaves environment tries to address this problem in two ways: by integrating a component browser, *Carlyle*, with Jacquard to permit users direct access to online component documentation; and by providing support for a wide variety of "annotations" that can be embedded directly into a weave. Carlyle supports several forms of component documentation, including a "manual page" that describes the component in detail, a "summary page" that provides a concise inventory of the function and the input/output expectations of the component, the source code (if any) for the component, and a sample weave that illustrates the use of the component. It is the last element that is often the most helpful, since developers can use the example weave as a starting point for the development of a specific application or component. With respect to managing large collections of components, Carlyle is wholly inadequate in many respects: it does not support indexing or search, does not assist users in discovering related components, and does not directly aid a user in understanding how a component fits into a larger picture of design and specification.

Weave annotations take many forms, including, but not restricted to, text, documents, diagrams, pictures and drawings, digital sound recordings, and digital movies. For example, the weave shown in Figure 3, comprising a portion of the mission processing for a

satellite ground station, contains annotations in four different media: text, documents, graphics, and digital sound. Annotations stay with the weave over its lifespan; in particular, weaves, along with their annotations, can be mailed from one site to another and directly executed, manipulated, and edited at the receiving site. In addition, the system structure is designed to accommodate new forms of annotation as they arise; for example, shortly after the appearance of Mosaic we began embedding Mosaic hypertext documents within weaves and are considering permitting a Mosaic URL (Uniform Resource Locator) as a legitimate annotation.

The annotations have proved to be surprisingly handy, allowing users to "decorate" a weave painlessly with large quantities of useful, related information. Some users employ Jacquard strictly as a documentation and system design tool, without ever intending to execute a weave. However, weaves can quickly become cluttered with annotations; and while we have introduced mechanisms for associating annotations with specific components, it is easy to lose track of the significance of the annotation and its "owner." In addition, annotations deserve the same forms of encapsulation, structure, and hierarchy employed by weaves, and although annotations can be nested in subweaves, it is clear that documentation and commentary deserve equal rank and weight.

## 2.3 System Browsing

- Weaves encourage the hierarchical composition of systems by allowing themselves to be encapsulated and used as components in higher-level weaves. The internal structure of such components is accessible in the same visual manner as the higher-level structures (Jacquard will open another worksheet containing the subweave comprising the component), and all of the direct manipulations described above can be applied to an encapsulated weave. However, users exploring deeply nested systems become irritated with the clutter of windows and quickly lose their place in the hierarchy. This problem is endemic to systems that use multiple windows to explore and display nested structures.

Another serious obstacle in the development of large systems is the problem of component discovery. Jacquard tries to address this problem by providing customizable *component trays*, organized collections of components specific to a particular domain. Users drag components from the trays onto the worksheet as they piece together a weave. Figure 2 shows a user constructing a weave designed to detect forest fires through spaceborne infrared telemetry. Appearing above the weave are two trays. The tray on the right (a default tray supplied by Jacquard) contains stock components, including an array of empty sockets with various degrees of fan-in and fan-out. The tray on the left is tailored to the domain of manipulating and analyzing satellite telemetry and contains numerous highly specialized components. New trays can be created and populated by users employing the editing facilities of Jacquard, and may contain a variety of objects including components, multimedia annotations, subtrays, and weaves. Although trays are certainly helpful, they also suffer from scaling problems. Trays are largely static organizations that are best suited to a small number ($< 100$) of components per tray; and while it easy to build specialized trays, the current environment does not help one discover the appropriate tray.

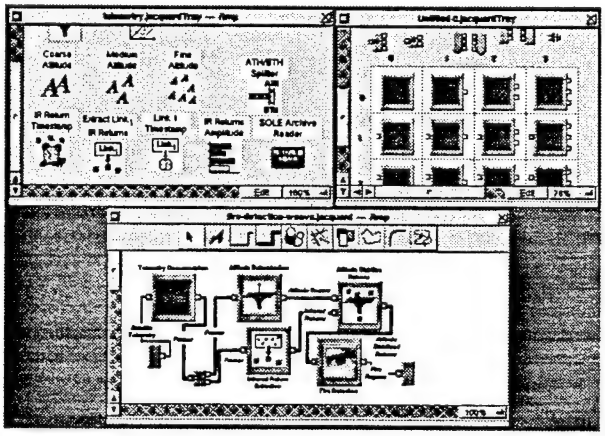


Figure 2: Using Jacquard to build a weave.

## 2.4 Summary

Weaves are a visual programming medium with the capability to support arbitrary object flows between components. Despite their considerable visual support for annotation, arbitrary object flows, and flexible hierarchical composition, they fall far short of the ideal visual software engineering environment. The following sections show how we are transitioning weaves from visual programming-in-the-small toward visual programming-in-the-large through the use of zooming and the explicit representation of the functionality of the components.

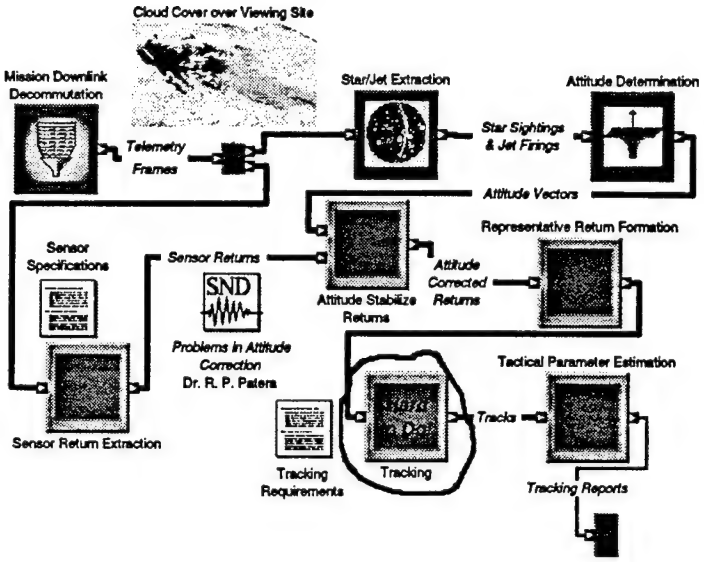



Figure 3: A weave for a satellite ground station that contains several annotations.

## 3 Zooming

Two fundamental issues in visual software engineering are visual browsing and visual access to relevant information. Zooming is a single, uniform visual mechanism, first introduced in Pad [15], that supports all forms of visual browsing and access.

We propose visualizing the information space of a software engineering environment as a $2\frac{1}{2}$D plane that not only supports the customary modes of planar navigation, but also supports *zooming*: as a user "approaches" an object it resolves itself into greater and greater detail. Conversely, as a user withdraws from an object, the detail gradually disappears until the object fades into the "distance." For example, as a user approaches a technical report, it may first appear as a document icon, then as a title and author, then as an abstract, and finally as a collection of the report's individual pages. This progression is illustrated from left to right in Figure 4.

Figure 5 shows how the stereo-tracker weave of Figure 1 might look in a visual software engineering environment where zooming is a fundamental visual metaphor. As we approach the weave the icons representing the components occupying the sockets fade away and are replaced with a high-level view of their fine structure. Drawing closer to any one component $C$ reveals its documentation, source code, examples of use, a catalogue of related components (for example, all those components that can either consume the objects output by $C$ or produce objects that are input by

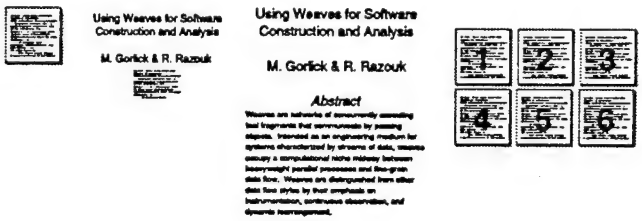



Figure 4: Successive views, from left to right, of a technical report as a user zooms in.

$C$), and an indication of $C$'s place in a greater family of components. Figure 6 shows such a closeup view of a typical component. Any element of this collection could be inspected in more detail by zooming closer to it. Finally, the socket on the far left of Figure 5 contains a component that is an encapsulated subweave. Zooming in on this portion of the weave exposes the detailed substructure shown in Figure 7, including a variety of annotations relevant to this subsystem.

This combination of planar navigation and zooming allows information to be presented at a scale and with a level of detail that are appropriate to the user's current focus and interests. A "distant" view suppresses detail, permitting only the most significant information (objects) to remain visible, while a "closeup" view brings forth increasingly detailed information (objects). Zooming is a visual metaphor that is particularly well suited to visual programming systems that support hierarchical composition.

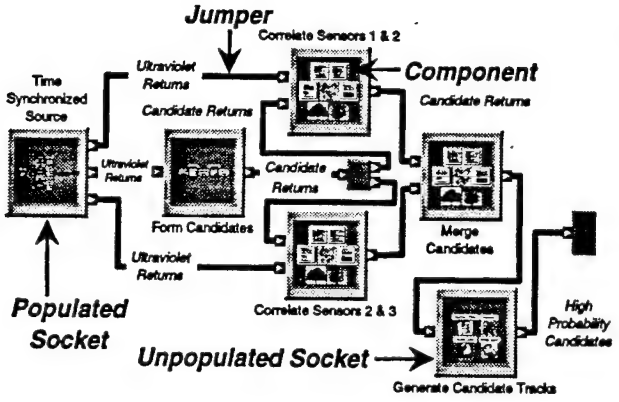



Figure 5: The appearance of a weave in a visual environment that supports zooming.

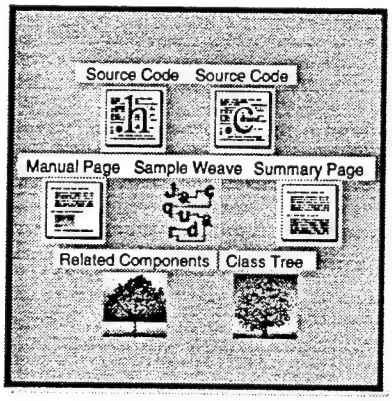



Figure 6: The view as one zooms into a weave component sitting inside a socket well.

## 4 Component Location and Retrieval

A key problem in constructing any large, complex system is locating the components that serve as its building blocks. This is especially important in the weaves environment, since its "notion" of visual programming corresponds to locating and interlinking components. We first briefly review some of the mechanisms proposed previously for component search and retrieval, and then detail how zooming, augmented with a classification knowledge base, can solve this problem.

While it is relatively straightforward to provide *purely* visual mechanisms for locating components, they don't scale. The component tray metaphor used in the current environment works well for constructing simple systems from small sets of components, but

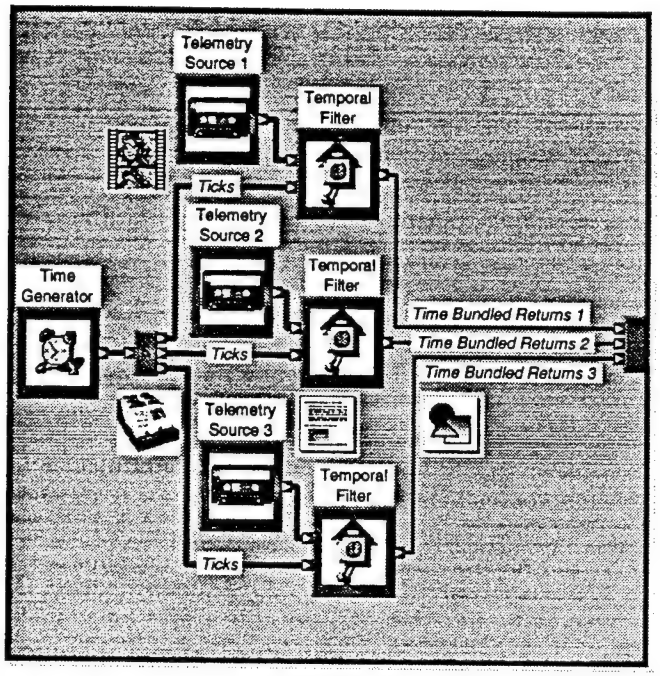



Figure 7: The view as one zooms into a subweave sitting inside a socket well.

breaks down when users must hunt through hundreds of trays, each containing hundreds of components.

In contrast, there are many *nonvisual* component-retrieval mechanisms designed to handle large component libraries, including the following:

**Keywords** Every component is described by a set of keywords. Users supply keywords and the system retrieves those components with matching keywords [7].

**Faceted classification** Every component is categorized according to different facets (actions and objects) of a domain model. Users specify the desired action and object and the system retrieves the matching components [6, 17].

**Semantic nets** Every component is linked to those concepts relevant to it. Users provide the names of relevant concepts and the system retrieves the most closely connected components [14].

**Subsumption** Every component's function is described in terms of an object-oriented domain model. Users provide detailed type constraints on actions and objects, and the system uses subsumption to retrieve components meeting those constraints [4].

It is straightforward to extend visual programming environments to use one of these component retrieval methods, but doing so has several significant drawbacks. First, these methods tend not to be visually oriented and require one learn a specialized retrieval language. Second, they treat component retrieval as a separate task, distinct from programming, this forces the user to shift repeatedly between programming and locating components. It seems undesirable to make a user master a special-purpose tool for locating program components when this tool differs substantially from the other visual programming tools.

Thus, there is a tradeoff: visual approaches to locating components don't scale, but nonvisual approaches are difficult to learn and use. In addition, both approaches have another, more subtle drawback: the user must shift from a visual task to the task of locating components and then use some search mechanism to find them.

We are currently extending the weaves environment with a mechanism that supports component reuse while avoiding these drawbacks. Here component discovery is performed visually, in a way that both scales and is completely integrated with the task of constructing the system. In particular, as a user constructs a weave, the system treats each user action (such as linking two sockets or annotating a connection) as an incremental visual specification of the weave's behavior. The system automatically notes which components meet the current specifications and makes these components visually accessible to the user.

For example, imagine a user assembling the weave shown in Figure 1. If the user creates a socket and attaches to it an input jumper for `Ultraviolet-Return` objects, the system will automatically determine the set of components that can accept a stream of `Ultraviolet-Return` objects as input. If the user later supplies the socket with an output jumper for `Candidate-Returns`, the system will then automatically narrow down this set to those components that can also produce `Candidate-Returns` as output.

To support this behavior, we need a formal mechanism for describing each component's behavior. We are using CLASSIC [1], a classification-based knowledge representation that supports hierarchically organized concepts and instances (much like an object-oriented database), automatic classification (the ability to determine where a new instance should be located in the knowledge base), and subsumption-based retrieval (the operation of retrieving all knowledge-base entities that meet the specified type constraints). The basic idea is that all components are represented in terms of type constraints on their inputs, outputs, and their overall function. As the user visually attaches jumpers sockets and specifies their types, these are interpreted as type constraints on the socket. Subsumption is then used to locate those components that meet the socket's constraints.

One problem with this approach is in determining how to display the large number of components that could potentially correspond to a loosely constrained socket (say, a socket constrained only to take a stream of `Things` as input and produce a stream of `Things` as output). This display issue is crucial, as the set of relevant components is constantly being recomputed and redisplayed whenever the user modifies an unpopulated socket. Our solution to this problem uses zooming, where zooming into an unpopulated socket reveals all the components that could populate it, given its input and output constraints. When there are few relevant components, zooming can provide a complete view of all of them, and when there are many possibilities, these components can be organized and displayed hierarchically, so that further zooming reveals more and more specific components (that is, components that deal with particular subclasses of the input and output objects that the user has specified for the socket). In either case, the user can then simply select one of these components to populate the socket. This approach is illustrated in Figure 8, where, as a user zooms into the unpopulated socket shown previously in Figure 1, the collection of components for candidate track generation is revealed. The key benefit of this approach is that the user need not ever explicitly search the space of all components for those that might be relevant. Instead, the system constantly maintains and displays the subset of relevant components.

## 5 Handling User Errors Visually

When constructing complex systems, users often err in how they connect modules together. This is especially true when novice programmers are visually assembling complex systems. In the weaves environment, these errors include linking together inappropriate sockets and filling a socket with an inappropriate component. Most visual programming systems handle errors in the obvious way: they simply report the error and let the user find and fix any problems. A more appealing approach, however, is to try to prevent these errors from occurring and to address their root causes when they do occur.

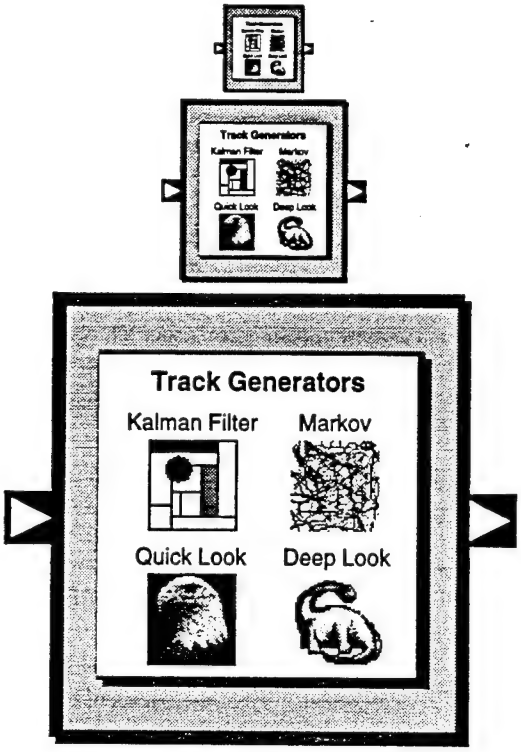



Figure 8: Successive views of an empty socket when zooming is used to discover components.

Many systems, including the weaves environment, try to reduce the frequency of errors by providing online documentation for each component, along with examples of its use. Our planned extensions to the weaves environment go beyond that by making accessible to a user only those components that are suitable for a given socket, thereby decreasing the chance that a user will fill a socket with an inappropriate component.

Despite these efforts, however, users still make mistakes. We intend to treat these mistakes as special types of requests. For example, one common user error is to link a socket producing objects of type $X$ to another socket expecting objects of type $Y$. The usual reason for this error is a user's belief that $X$s can go anywhere $Y$s can go (which is only true if $Y$ subsumes $X$). Presenting an error message that $X$ is not a subclass of $Y$ simply forces the user to find some means of transmuting $X$s into $Y$s. Thus, the system can aid the user by treating this error as a request to splice in a socket that can transmute an $X$ into a $Y$; the system responds by inserting a new socket and filling it with a suitable transmuting component. If there are multiple components, these are presented within the socket, so that the user can use zooming to examine them and choose which one to use. On the other hand, if there are no such components, the system can attempt to synthesize a "bridging weave" capable of performing the appropriate transformation. (This is tractable, as a weave that can transmute objects from one type to another reduces to the problem of finding a path between two nodes in a type graph.) Other common errors, such as a user populating a socket with a component that does not conform to the socket's input and output requirements, can be handled in a similar way.

The primary drawback to treating user errors as requests is that the user may have simply made a mistake. For example, the user may have meant to link sockets A and B, but mistook a nearby socket C for B and linked A to C instead. In this case, the system is wasting time trying to find an intermediary way to connect A to C. Given our visual editor, however, it is easy for the user to delete any automatically created connection and reconnect the socket to the right place. An alternative would be to ask the user before attempting to find this connection. We are exploring both mechanisms for dealing with perceived user mistakes. The most important point, however, is not merely to point out the mistakes users make when assembling complex systems, but to also help users correct those mistakes.

## 6 Representation of Component Function

The goals of supporting component location and retrieval and treating user errors as requests require a detailed representation of the function of each component. Our approach to representation is based on the scheme used by the LASSIE [4] system to represent the function of modules in telephone switching software. LASSIE was designed to retrieve C source modules based on conceptual descriptions of the domain actions that they performed, such as "allocating a trunk line" or "flashing a console light."

Like LASSIE, our representation for components relies on a classification-based description of domain objects and the input/output behavior of components (which can be formally represented in a terminological logic-based framework like that of [1]). Figure 9 shows an portion of a domain model used to describe the components of the weave in Figure 1. In this particular case, the domain objects include items such as

Sensor-Returns, Tracks, Attitudes, and so on. Similarly, the domain actions correspond to manipulations of the domain objects such as filtering sensor returns, transmuting telemetry frames to sensor returns, and so on.

All entries in the domain model have attributes. Object attributes correspond to subcomponents of the object and differ according to the object's type. A Sensor-Return, for example, has the attributes of location, platform, and spectral-range, while an Attitude has the attributes of frame of reference, accuracy, and coordinates. Components have input and output attributes that describe the types of objects they require as inputs and produce as outputs (as well as having constraints on particular object attributes) For example, the component

High-Intensity.Ultraviolet.Sensor-Return.Filter

requires an Ultraviolet.Sensor-Return as input and produces a High-Intensity.Ultraviolet.Sensor-Return as output, and the component

Time-Synchronized.Sensor-Return.Generator

produces a triple of Sensor-Returns as output, with each return corresponding to an observation from a different spaceborne platform.

Given this domain model, we represent the behavior of each component in terms of the types of objects that it consumes and produces and the type of action that it performs. Figure 10 shows a detailed description of some of the component domain model.

This classification-based domain model supports our locating appropriate components by specifying their function in terms of type constraints on its action and its input and output objects. These queries are resolved by *subsumption*, the operation of finding all items in the knowledge base that meet the specified type constraints. In essence, the user visually provides type constraints and we use subsumption to locate the components meeting those constraints.

## 7 Summary

Just as programming-in-the-large is substantially different from programming-in-the-small, so is visual programming-in-the-large greatly different from visual programming-in-the-small. Unfortunately, most current visual programming systems are focused on programming-in-the-small and do not adequately address the issues that arise in visual software engineering — trying to construct sizable, complex systems assembled from thousands of components.

Figure 9: A Portion of the Satellite Telemetry and Tracking Domain Model for Weaves

- Object
  - Sensor-Return
    - Ultraviolet
      - Low-Intensity
      - Medium-Intensity
      - High-Intensity
    - Radar
  - Track
    - Low-Confidence
    - Medium-Confidence
    - High-Confidence
    - Confirmed
  - Attitude
  - Sighting
    - Earth
    - Star
    - Sun
  - Telemetry-Frame
    - GOES-Telemetry-Frame
    - NOAA-Telemetry-Frame
  - Pair
    - Sensor-Return
- Component
  - Generator
    - Track
    - Sensor-Return
      - Time-Synchronized
  - Filter
    - Sensor-Return
      - Ultraviolet
        - High-Intensity
  - Transformer
    - Telemetry-Frame-To-Sensor-Return
    - Star-Sighting-To-Attitude
  - Composition
    - Pair
      - Sensor-Return

Figure 10: A Portion of the Component Model for Weaves

High-Intensity.Ultraviolet.Sensor-Return.Filter
  **Input:** Ultraviolet.Sensor-Return
  **Output:** High-Intensity.Ultraviolet.Sensor-Return
Time-Synchronzied.Sensor-Return.Generator
  **Output:**
    (Sensor-Return [platform=#1]
    Sensor-Return [platform=#2]
    Sensor-Return [platform=#3])
Sensor-Return.Pair.Composition
  **Input:**
    (Sensor-Return [platform=?X]
    Sensor-Return [platform=?Y])
  **Output:** Sensor-Return.Pair

This paper has described an existing visual programming system and a set of proposed extensions that address some of the outstanding issues of visual software engineering. The primary requirement underlying our work is the need to support the construction of complex systems that have thousands of interconnected components and tens of thousands of related information artifacts. This effort has led us to develop a set of principles that we believe are relevant to anyone constructing systems for visual programming-in-the-large:

- All tasks related to visual programming, such as component discovery, should be *visually integrated* with the task of constructing visual programs.
- All relevant "software engineering" information (such as requirements, specifications, design, documentation, and test data sets) should be accessible in a single, unified visual manner.
- All visual programming environments must provide visual mechanisms for examining visual programs at different levels of detail.
- All apparent errors made while composing programs visually should be interpreted as requests for information or additional components, not as mistakes. It is important to minimize the number of user actions that are treated as errors.
- An incomplete visual program should be considered as a partial specification.

It has been a long and painful transition from programming-in-the-small to programming-in-the-large. Avoiding making the same mistakes when constructing systems for visual programming-in-the-large means putting into visual practice the lessons learned from this earlier transition.

## References


[1] R. J. Brachman et al. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, chapter Living with CLASSIC: When and How to Use a KL-ONE-like Language, pages 401–456. Morgan Kaufman, 1992.

[2] K. Brade, M. Guzdial, M. Steckel, and E. Soloway. Whorf: A visualization tool for software maintenance. In *Proceedings of the 1992 IEEE Workshop on Visual Languages*, pages 148–154, Seattle, Washington, September 1992. IEEE Computer Society Press.

[3] Frank DeRemer and Hans H. Kron. Programming-in-the-large versus programing-in-the-small. *IEEE Transactions on Software Engineering*, SE-2(2):80–86, June 1976.

[4] Premkumar Devanbu, Ronald J. Bachman, Peter G. Selfridge, and Bruce W. Ballard. LaSSIE: A knowledge-based software information system. *Communications of the ACM*, 34(5):34–49, May 1991.

[5] R. A. Earnshaw and N. Wiseman. *An Introductory Guide To Scientific Visualization*, chapter 8. Springer-Verlag, 1992.

[6] D. W. Embley and S. N. Woodfield. A knowledge-structure for reusing abstract data types. In *Proceedings of the 9th International Conference on Software Engineering*. IEEE Computer Society Press, 1987.

[7] W. B. Frakes and B. A. Nehmeh. An information system for software reuse. In *Proceedings of the Tenth Minnowbrook Workshop on Software Reuse*, pages 142–151, 1987.

[8] Michael M. Gorlick and Rami R. Razouk. Using weaves for software construction and analysis. In *Proceedings of the 13th International Conference on Software Engineering*, pages 23–34, Austin, Texas, May 1991. IEEE Computer Society Press.

[9] Paul E. Haeberli. ConMan: A visual programming language for interactive graphics. *Computer Graphics*, 22(4):103–111, August 1988.

[10] R. Kadia. Issues encountered in building a flexible software development environment: Lessons learned from the Arcadia project. In *Proceeedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments*, pages 169–180, Tyson's Corner, Virginia, December 1992.

[11] M. Kado, M. Hirakawa, and T. Ichikawa. HI-VISUAL for hierarchical development of large programs. In *Proceedings of the 1992 IEEE Workshop on Visual Languages*, pages 48–55, Seattle, Washington, September 1992. IEEE Computer Society Press.

[12] David W. McIntyre and Ephraim P. Glinert. Visual tools for generating iconic programming environments. In *Proceedings of the 1992 IEEE Symposium on Visual Languages*, pages 162–168, Seattle, Washington, September 1992. IEEE Computer Society Press.

[13] Andreas L. Opdahl. Structured analysis, structured design, visual programming. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*, pages 292–297, Bergen, Norway, August 1993.

[14] E. Ostertag and J. A. Hendler. AIRS: An AI-based Ada reuse system. Technical Report CS-TR-2197, Computer Science Center, University of Maryland, 1987.

[15] Ken Perlin and David Fox. Pad: An alternative approach to the computer interface. In *SIGGRAPH 93 Conference Proceedings*, pages 57–64, Anaheim, California, August 1993. ACM SIGRAPH, ACM Press.

[16] Dewayne E. Perry. The Inscape environment. In *Proceedings of the 11th International Conference on Software Engineering*, pages 2–12, Pittsburgh, Pennsylvania, May 1989. IEEE Computer Society Press.

[17] R. Preito-Diaz and P. Freeman. Classifying software for reusability. *IEEE Software*, 4(1):6–16, January 1987.

[18] S. P. Reiss. A framework for abstract 3D visualization. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*, pages 108–115, Bergen, Norway, August 1993. IEEE Computer Society Press.

[19] Craig Upson et al. The application visualization system. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

# A Process-centered Methodology for Engineering of Complex Systems

Azad M. Madni*

## 1. Introduction

The Engineering of Complex Systems (ECS) is a complex process that involves multiple design iterations and tradeoffs at multiple levels of abstraction. Yet the viewpoint of process remains implicit in the systems engineering life cycle, methodologies, and tools. There are several reasons why this separation continues to exist between two complementary, and ostensibly synergistic disciplines. First, process engineering technology is relatively new on the market. Second, process engineering advocates who have historically focused on software engineering are just beginning to take a hard look at systems engineering [Madni, 1990]. Third, the return on investment of inserting a process viewpoint within systems engineering has not been quantified. Finally, the interface between process engineering technology and systems engineering has not been defined. Compounding the problem is the fact that the complex systems engineering process varies with several factors. These factors include: 1) required use of legacy in the project; 2) variability due to differences in personnel perspectives, skill level, and domain knowledge; 3) differences in system design objectives (e.g., design-to-cost, design for reliability, some combination of design-for-X); 4) differences in "best practices" approach within an organization. With such variability, and without an explicit characterization of objectives, assumptions, and "flows" associated with ECS process, process benchmarking becomes intractable and process improvement becomes an elusive goal. This recognition provides the impetus for a process-centered approach to ECS. In this paper, we first establish the link between process engineering technology and systems engineering, and then present the system concept for a process-centered ECS methodology. We discuss the advantages of a process-centered methodology followed by a description of an architecture for implementing a process-centered approach to ECS. We conclude by presenting future R&D directions to resolve key issues involved in realizing a process-centered approach to ECS.

## 2. Process-centered ECS Approach

The process-centered ECS approach introduces a process viewpoint in complex systems engineering. The addition of a process view to the system view makes concurrent engineering possible. When the system perspective (i.e., the what) and the process perspective (i.e., the how) are put together, the integrated model becomes capable of supporting truly concurrent development (Figure 1). In fact, process engineering technology is well-suited to aiding, automating, guiding, and managing the ECS process. Specifically, the ECS process management function can guide, track, monitor, and measure the ongoing systems engineering processes. For example, it can guide the user in which tool to use, what tradeoffs to consider, which team members to inform about the current status, and what the distribution list is for a specific interim product.

## 3. Advantages of a Process-centered ECS Methodology

To date, the ECS community has developed several key methods and tools (e.g., DESTINATION, SOHAR, SMERFS, SES, RTO.k) that aid or automate individual ECS functions. These tools have demonstrably increased the productivity and consistency of individual ECS activities (e.g., testing, reliability prediction). A contrasting viewpoint is offered by process-centered ECS environments that focus on global optimization and tool integration (Table 1). Most importantly, a process-centered perspective provides valuable insights into improving the overall ECS process.

---

* Azad M. Madni is with Intelligent Systems Technology, Inc., Los Angeles, CA 90064.

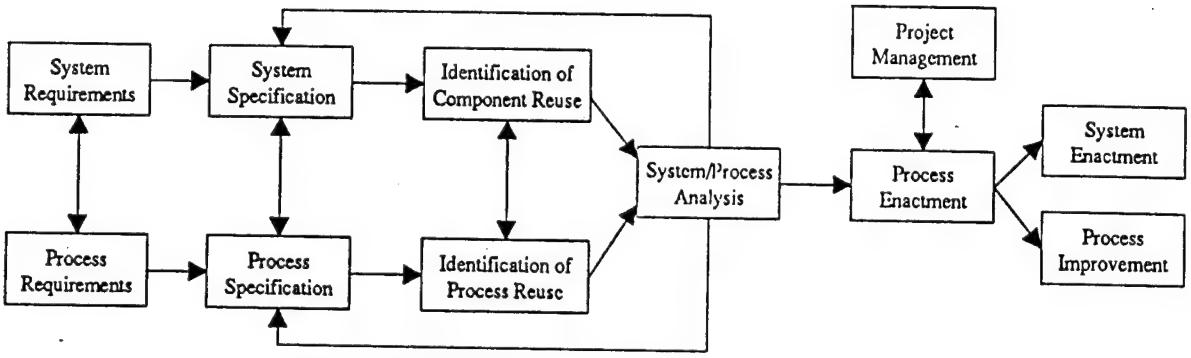



Figure 1. Process-centered ECS Approach

Table 1.
Traditional Environment vs. Process-centered Environment

| Comparison Criteria | Traditional ECS Environment | Process-centered ECS Environment |
|---|---|---|
| Tool invocation/ termination condition | user-determined | process-guided with user override |
| Tool integration | software integration issue | process integration issue |
| Guidance | none | process-based |
| Metrics | product/system | system, process, organization, tools |
| Design iterations | unmodeled and informal | modeled and tracked |
| Methodology | tool-driven | process-centered |
| Representation | descriptive generally; incomplete semantics | executable; semantically complete |
| Product quality | high variability; conformance to specification compromised | repeatable, measurable, traceable to process steps |
| Continuous improvement | an additional concern | intrinsic to paradigm |
| Global optimization | not a principal concern | principal concern |

## 4. Requirements for a Process-centered ECS Environment

The ECS process is characterized by several factors that have a direct impact on the creation of a process-centered ECS environment. Table 2 presents some of the salient characteristics of complex systems engineering process.

Table 2.
Characteristics of ECS Process

- Multiple disciplines and multiple functions involved in the complex system life cycle.
- Geographically dispersed "virtual teams", "virtual corporation."
- Hardware, software, humanware components.
- Multiple operational modes and levels of automation.
- Manual, automated, and shared human-machine processes.
- Multiple design iterations particularly in early stages are commonplace.
- Multiple interfaces and handoffs between different engineering disciplines.
- External interfaces with subcontractors, vendors, suppliers, and customers.
- Heterogeneous legacy tools and data bases that support individual ECS functions and activities.

The requirements for a process-centered ECS environment can be derived from the characteristics of the ECS process. Table 3 presents these requirements.

Table 3.
Requirements for a Process-centered ECS Environment

- Timely feedback across disciplines and across organizational boundaries is essential for agility.
- Evolutionary architecture and open interfaces to accommodate new requirements/technology options.
- Need to integrate heterogeneous tools as well as legacy systems and data bases.
- Tailor a reference process model to contractor-specific processes, and organizational constraints.
- Enact and track multiple design iterations and concurrent instances of ECS subprocesses.
- Need to "mix and match" multiple methods, techniques, and tools.
- Manage interfaces and handoffs between different disciplines.
- Manage interfaces with subcontractors, vendors, suppliers, and customers.

## 5. Process-Centered Architecture for ECS

The process-centered methodology focuses on providing automated process support and guidance during development. These capabilities can be made available over selected wide-area networks (e.g., Internet, EINet, Mosaic). At the heart of the methodology is an object-oriented, integrated, multi-disciplinary model of the complex systems. The object-oriented model serves as a logically centralized data and information repository for the design and representation of a complex system. It can be implemented on a fully distributed network of relational data bases, object-oriented data bases, and knowledge bases. A corresponding process support framework can be built on top of the object-oriented model to support the definition, simulation, enactment, and management of the ECS process. A scalable 6-layered architecture for the process-centered ECS methodology, is shown in Figure 2.

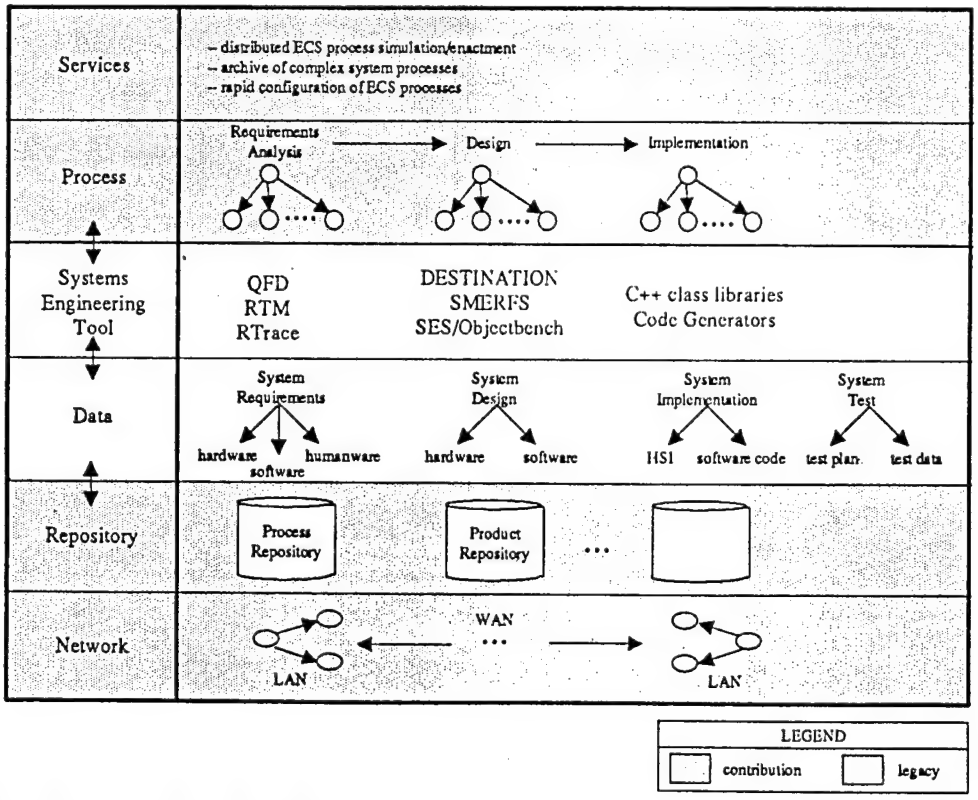



Figure 2. Process-centered Scalable 6-Layered Architecture

The scalable 6-layered ECS architecture is capable of supporting the full ECS life cycle while providing a process integration framework for new and legacy tools. The existing systems engineering tools support the two middle layers of the architecture. The data layer stores and provides access to the structure as well as the content of system data that are used in ECS. Examples of the data include: user requirements, system design, tradeoff analysis, and executable code for software subsystems. Currently, there are COTS tools, such as relational data bases, object-oriented data bases, and other types of special purpose data repository systems to support data repository requirements for ECS. The ECS tool layer provides software systems that support individual ECS tasks, such as Requirements Analysis, Design, and System Implementation. Both COTS and Navy-supported ECS tools can be integrated into this process-centered ECS architecture with minimal cost impact

The other four new layers in the architecture are new to ECS and unique to the proposed methodology. The two lowest layers, the network and repository layers, provide distributed repository and access of the object-oriented model and the development data associated with the model. These two layers can be implemented through customizing and enhancing COTS tools. The two highest layers, i.e., the process and service layers, provide network-available process support capabilities to remote users.

## 6. ECS Process Representation Requirements

Most process tools have incomplete design representation. This inadequacy manifests itself when one attempts to analyze, manage, track, and measure the process, complex or not. The requirements for a complete representation of the ECS process are presented in Table 4.

Table 4.
ECS Process Representation Requirements

- Explicit Executable Model
  - -- explicit modeling of hardware, software, and humanware
  - -- model "execution": generates timeline and resource utilization profiles
- "Flow" Modeling
  - -- model control flow, data flow, and entity types, e.g., process
  - -- modeling of sequential, parallel, conditional, iterative processes and multi-process instantiation
  - -- hard, semi-hard and soft temporal constraints
- Process Analysis Support
  - -- basic (i.e., syntactic) checking
  - -- simulation and predictive timeline, resource, and cost analyses
  - -- in-situ measurement analysis
- Process Tailoring
  - -- to contractor-specific processes, policies, and procedures
- Process Use Support
  - -- process enactment
  - -- process management, i.e., monitoring, scheduling, querying, responding, measuring
  - -- repository
- Process Visualization
  - -- different user perspectives (manager, designer) and graph-based graphical interface (e.g., IDEF)
  - -- different entity types
- Usability
  - -- ease-of-use, scalability, life cycle support

7. Process Integration

One of the major advantages of a process-centered approach is its ability to integrate existing ECS tools through an innovative technique called process integration [Mi and Scacchi, 1992; Madni, 1990]. The key notion behind process integration is to embed the necessary physical information about tool integration (such as physical location of the tool and transferring of necessary parameters) into the development tasks that use the tool in the ECS process and then to invoke the tool as soon as the development tasks are enacted. This technique effectively hides all physical information of tool invocation from potential users, who invariably are systems engineers. Through this notion of process integration, the process-centered ECS methodology can provide only the logical information that is relevant to the ECS developers. The process integration concept is illustrated in the following paragraph using DESTINATION as an example of a tool that has been selected for integration.

The DESTINATION methodology and toolkit provides for Design Structuring, Resource Allocation, Design Evaluation and Optimization. DESTINATION is capable of supporting various design methodologies (e.g., object-oriented design, structured design, task structuring, partitioning) but only implicitly, i.e., DESTINATION users have to take initiative in following a particular methodology and selecting tools accordingly. The process-centered ECS approach, on the other hand, call for an explicit description of the ECS methodology. This approach has several advantages. First, users can choose and customize a design methodology for a complex system. Second, users can load the chosen design methodology into the process enactment engine and start the ECS process. Along the way, each of the design tasks, such as design generation, design evaluation, tradeoff analysis, get enacted in turn in accord with the selected methodology. Furthermore, as a design engineering task is enacted, specific DESTINATION tools (e.g., design structuring or partitioning) can be invoked on the appropriate system design data. In this way, data correctness and consistency as well as proper tool selection is ensured. In the event that DESTINATION does not support a particular design task, the process-centered ECS approach will be capable of invoking a third party system design tool and performing the necessary conversion and transformation of data formats.

8. Future Directions

While process-centered methodologies hold great promise for ECS, several key developments have to occur and key issues resolved before the full impact of process-centered environment is felt in ECS. First and foremost, is the need for a consistent and integrated representation of the system and its subsystems, their functions and behaviors, their development processes, as well as special considerations such as fault tolerance and security. Second, is the need for a distributed process management system capable of orchestrating and managing human, equipment, and monetary resources as well as artifacts produced and consumed during the ECS process. Third, is the basis for tool integration from a process-driven perspective. Fourth, is the integration of new software with legacy systems. When these key concerns have been successfully addressed, process-centered environment capable of providing automated guidance will become a reality.

References

Madni, A.M., Freedy, A., Estrin, G.R., and Melkanoff, M. Concurrent Engineering Workstation for Multi-Chip Module Product Development Process. Invited Paper presented at CALS & CE Washington '91 Conference and Exposition, Washington, D.C., June 1991.

Madni, A.M. A Conceptual Framework and Enabling Technologies for Computer-aided Concurrent Engineering (CACE). Plenary address & Invited Paper, Second International Conference on Human Aspects of Advanced Manufacturing and Hybrid Automation, August 12-16, 1990.

Madni, A.M. HUMANE: A Knowledge-Based Simulation Environment for Human-Machine Function Allocation. Proc. of IEEE National Aerospace & Electronics Conference, 1988.

Mi, P. and Scacchi, W. "Process Integration in CASE." IEEE Software, 1992.

# A Methodology Framework for Optimal Design of Real-Time Dependable Computer Systems

K. H. (Kane) Kim
University of California, Irvine

Mary Denz & Thomas Lawrence
USAF Rome Laboratory

Cuong Nguyen & Richard Scalzo
USN Naval Surface Warfare Center

## Abstract

Resource allocation in complex real-time (RT) computer systems cannot be adequately handled by straightforward extensions of the CPU and peripheral device scheduling techniques used in non-RT computer systems. In this paper, we attempt to lay out an idealistic methodology framework for optimal resource allocation in complex RT computer systems, i.e., optimal design of complex RT computer systems. One useful property of the optimal design methodology framework presented is the illumination of the separation between the design activities that are amenable to automation and those activities which must be handled by the system engineer. The fundamental notion of timed value accuracy is formalized first and then the quantitative specification by the system engineer of the relationship between the loss in the timed value accuracy of each output action and the damage to the application mission is adopted as a key ingredient in this idealistic optimal design methodology. The importance of treating fault tolerance requirements as an integral part of the initial system requirements and the need for further accumulation of knowledge on the resource requirements and the effectiveness of RT fault tolerance techniques, are substantiated. The major challenges posed to the system designer in effectively practicing the proposed optimization methodology, are also discussed.

## 1. Introduction

Complex real-time (RT) computer systems consist of many multi-purpose modules sharing the workload for providing a variety of time-critical service functions to achieve application objectives. Therefore, both static (design-time) and dynamic (run-time) mapping of various computation segments to execution machine components in such systems have been treated as a major research and development (R&D) issue for at least two decades [Hal94, How93]. Complex RT systems are invariably of the distributed system type and the percentage of them containing parallel computers as their components has been steadily growing. The typical service actions of a RT computer system are combinations of
(a) outputting control values to devices in the application environments and
(b) storing newly computed values into a database which is shared by users outside the control domain of the RT computer system.

However, the techniques for both static and dynamic resource allocation in such systems have been advancing rather slowly. The main reason is because resource allocation in complex RT computer systems cannot be adequately handled by mechanical or straightforward extensions of the CPU and peripheral device scheduling techniques used in non-RT computer systems. Nor can they be adequately handled by straightforward extensions of the simple-structure periodic task scheduling techniques developed for centralized RT computer systems. It has simply taken long for the R&D community to come to this realization.

To be more specific, past experiences of the R&D community brought the following realization.

(1) In many safety-critical applications, it is too dangerous to accept a system design based solely on an *average performance* measure. The *worst-case* (i.e., the worst among the cases worth considering) *performance* measure is often more important.

(2) Scheduling each local resource or managing each *service infrastructure* (e.g., communication, shared data storage, etc.) to handle a maximal number of requests from clients of different importance, has been studied for long but each resource domain has been studied independent of

others. Independently devised resource managers do not integrate well among themselves or they even conflict with one another because they were designed to pursue different optimization objectives. Such a group of resource managers cannot create a *globally optimal resource allocation*, i.e., a situation where available resources are used the most efficiently to meet the objectives of the applications.

(3) Viewing several shared resources as infrastructures each of which serves an unpredictable mix of clients has encouraged the conceptualization of multiple resource domains subject to uncoordinated or loosely coordinated control but has not encouraged RT system engineers to take a global view of the system-wide impacts of the actions by each individual resource manager.

(4) In challenging RT application environments each multi-purpose execution resource is subject to experiencing internal faults with non-negligible probabilities and is made to shift dynamically among multiple operating modes with changing responsibilities. If an execution component fails permanently, then a system-wide reconfiguration must take place. A client may be designed such that if it finds itself to be in fault, then it attempts to recover from the fault by the means available in its sphere of control or by relying on a party of higher authority. Reconfiguration and recovery actions represent major cases of resource management. Optimizing these parts of the system resource management represents a much more important subject for research at this time and advances in this area will have much bigger impact in the improvement of the ability to produce ultra-reliable cost-effective RT computer systems.

(5) Integrated optimization of both static allocation and dynamic allocation has remained a distant goal. Since the resource allocation problem is equivalent to the problem of making some static allocations (which are design activities) plus designing some dynamic allocation algorithms, it is essentially a design problem. Therefore, realizing an optimal resource allocation means an optimal design of a system.

In this paper, we attempt to lay out an idealistic methodology framework for optimal resource allocation in complex RT computer systems, i.e., optimal design of complex RT computer systems. We believe that establishing an idealistic methodology framework is important because it enables system engineers to understand when they have to start deviating from or approximating the idealistic methodology framework in their practices due to the impossibility of obtaining accurate information on interplay among a myriad of parameters characterizing a complex system. Availability of an idealistic methodology framework will thus encourage system engineers to narrow down the portions of a system engineering process in which heuristic decisions must be made. With these understanding, system engineers are more likely to produce system designs closer to truly optimal designs than what the engineers would produce without such understanding and without a global view of resource needs.

Idealistic optimization goals are defined first in Section 2. The requirements such as processing throughput, response time, fault tolerance, and both development cost and operating cost constraints, are addressed. Section 3 then provides discussions on some challenges faced in pursuing idealistic optimization goals and making sensible compromises.

## 2. Idealistic optimization goal

Figure 1 depicts the problem space in which major factors to be considered in optimal design are highlighted. Formal definitions of the important factors are provided in this section.

### 2.1 Real-time functional requirements and accuracy of output

RT computer systems must perform time-critical service functions. Ideally, a RT computer system must take every service action "accurately". This means that every output from a RT computer system must be accurate in both the time dimension and the logical value dimension.

**Definition**:
(D1) The accuracy of an output of a RT computer system is the closeness of both time and logical value attributes of the output to the time and logical value attributes of the desired output. The value of an output of a RT computer system is thus a two-dimensional value called a timed logical value.
(D2) The temporal accuracy of a data transmission or storing action in a RT computer system is the closeness of the time attribute of the action to the time attribute of the desired action.
(D3) The logical value accuracy of a data transmission or storing action in a RT computer

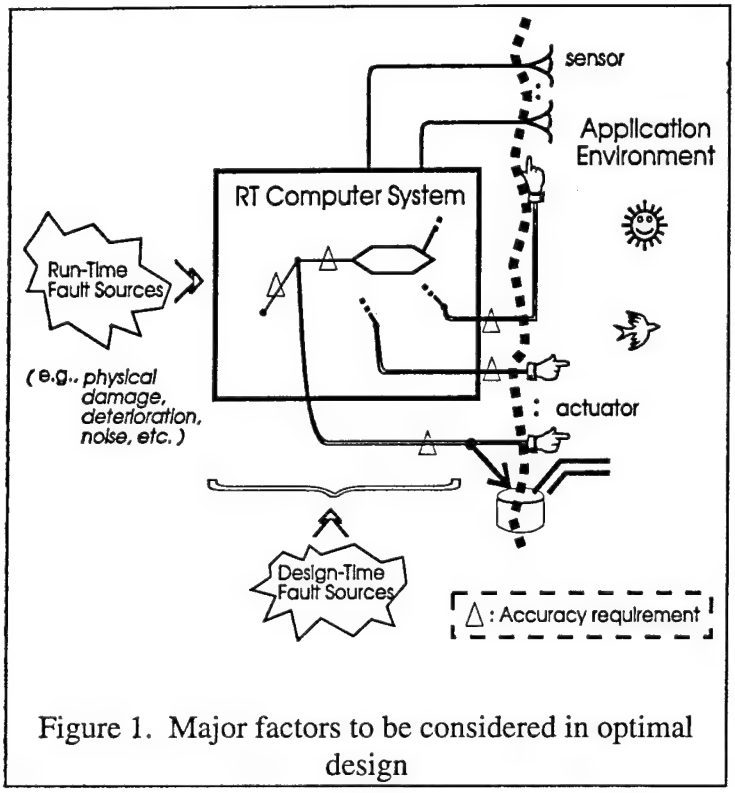



Figure 1. Major factors to be considered in optimal design

system is the closeness of the logical value attribute of the data involved in the action to the logical value attribute of the desired action. ■

When we measure the logical value accuracy of an output or a data storing action, the quality of measurement depends on the numerical precision established. Similarly, the quality of measurement of the temporal accuracy depends on the RT clock precision established. The numerical precision and the clock precision together can be referred to as the timed value metric precision.

**Definition**:
(D4) An ideal RT computer system is one which yields the highest possible degree of timed value metric precision and produces every output with the highest measurable degree of (timed value) accuracy. ■

In practice, complex RT computer systems are rarely immune from exhibiting inaccurate output. Examples of the manifestations of inaccurate output are:
(1) Omission of a desired output, i.e., output with zero degree of temporal accuracy;
(2) Too late output, i.e., output with an unacceptably low degree of temporal accuracy;
(3) Unacceptable logical value, i.e., output with an unacceptably low degree of logical value accuracy.
These examples also imply that an important thing to pursue in practical design is the *acceptable degree of accuracy*, not the absolute (maximum) accuracy. It is often not easy to evaluate how much damage inaccurate output causes to the application. The difficulty of quantifying the damage is mainly due to the fact that the nature of the application often allows a certain degree of self-healing capabilities to be embedded in the RT computer system. Therefore, the question of *how critical each output action is* does not yield an easy quantification procedure. In general, it can be judged only heuristically by the system engineer.

In principle, the system engineer should produce *a system which never fails to meet the specified output accuracy requirements except when the circumstances which the engineer decided not to cover arise*. As mentioned in Section 1, it is an elusive goal in the current practice.

How does an unacceptably inaccurate output occur ? Figure 1 indicates that an unacceptably inaccurate output is often preceded by an inaccurate data transmission or storing action within the RT computer system. In other words, an unacceptably inaccurate output is often preceded by an inaccurate intermediate result. Therefore, a more general question is: how does accuracy loss occur ?

Component failures are obviously the major cause. In addition, one may say that inefficient dynamic resource allocation can lead to accuracy loss somewhere within the RT computer system. For example, when the execution resource requirements of a certain service function increase sharply at some points in run time, an inadequate allocation of resources to that function may occur and then cause accuracy loss incurred in the execution results of that function. However, in principle, a resource manager in such a system is a faulty component (a component inflicted by a design fault) unless the particular circumstances in which resource requirements jumped sharply were deliberately taken out of the design specification by the system engineer due to the negligible occurrence probability. Therefore, an important near-term challenge that the RT computer system R&D community is currently facing with is

*to establish a systematic design methodology for producing RT computer systems with the guarantee that the systems will never fail to meet the output accuracy requirements in the absence of component failures.*

A noteworthy point here is that meeting the specified output accuracy requirements is not always a sufficient condition for the system to meet the application objectives well. The specification of accurate output requirements must be designed properly and this is the responsibility of the system engineer. This specification activity is sometimes a great challenge as will be discussed in Section 3.

### 2.2 Component failures as major causes for loss of output accuracy and resulting benefit losses

As mentioned in the preceding section, component failures are the most serious causes for loss of output accuracy. Therefore, the most important research issue in the field of optimal design of complex RT computer systems is

*the resource allocation to maintain output accuracy in the presence of component failures.*

Component failures are often caused by run-time fault sources such as
(1) physical damage,
(2) material deterioration,
(3) electromagnetic noise,
(4) radiation, and
(5) temperature,
and sometimes by design faults. Since it is not possible to tackle all conceivable combinations of component failures, the system engineer must define the types of system faults (i.e., states of the system which has deviated from the specified normal states due to permanent or temporary malfunctioning of some components) to be tolerated.

An important point to note here is that this specification of fault tolerance requirements must be done together with the specification of basic RT functional requirements. That is,

*fault tolerance requirements must be an integral part of the initial system requirements.*

Therefore, the system engineer must choose appropriate system "designs" to meet the fault tolerance requirements and this activity involves both static allocation of some execution resources and design of some dynamic allocation algorithms. A comprehensive knowledge-base on the resource requirements and the effectiveness in output (or intermediate result) accuracy maintenance of each useful fault tolerance technique is thus needed. Such a knowledge-base has not been fully constructed yet.

For example, a single-processor recovery block scheme is to execute the primary algorithm of an application function first and then execute a result check function called an acceptance test function [Ran75]. If the acceptance test result is negative, then the processor rolls back and makes a retry with the same algorithm or an alternate algorithm if available. This backward recovery can be quite time-consuming and thus under a certain fault occurrence scenario worth considering a substantial loss of the timed value accuracy in an output of the application function can result. On the other hand, the distributed recovery block (DRB) scheme uses two or more processors but it effects fast forward recovery [Kim94]. Therefore, for the same fault occurrence scenario, the DRB scheme yields much lower accuracy loss in the timed value output of the application function. For implementation of each application function, the system engineer can thus choose between the single-processor recovery block scheme and the DRB scheme, depending upon the processor availability, the expected fault rate, and other factors. In fact, even during run time, the system can switch the execution mode between the single-processor backward recovery mode and the DRB mode as the processor availability changes [Kim92].

Given a set of service functions defined as the mission of a RT computer system under development, the basic required resource set can be defined as a minimum set of resources which collectively provide enough processing capacity to support the mission when the possibility of any component failure during the application life-time can be ruled out. Therefore, mobilizing any additional resources beyond the basic required resource set will return no benefit as long as no components in the basic set fail. However, component failure probabilities are non-negligible in practice and thus some extra resources not included in the basic required resource set must be incorporated into the RT computer system.

When a sequence of component losses causes the system resource condition to fall below the level to form a basic required resource set, a very difficult resource allocation problem is posed. For example, a choice may have to be made between sacrificing a certain service function completely, thereby

effectively cutting off the output channel for that function, and degrading the output accuracy of multiple service functions to some extent. In principle, such a trade-off must be made on the basis of a rigorous evaluation of which option will cause less damage to the application mission.

Each time an output action occurs with degraded accuracy, some damage may occur to the application mission. The damage is called the benefit loss.

**Definition:**
(D5) When the accuracy loss $\Delta_v$ in an output action v occurs, the benefit loss $BL(\Delta_v)$ results. The mapping of every accuracy loss event to a consequent benefit loss value is called the benefit loss function. ■

The determination of $BL(\Delta_v)$ for every output action v in a RT computer system is the responsibility of the system engineer. For some highly critical output actions whose failure directly leads to the application mission failure, it will be easy to determine the magnitude of the benefit loss which is the maximum in the (benefit loss) value range adopted. For other output actions, the determination of the benefit loss must involve heuristic judgment by the system engineer. In a sense, the benefit loss function for a particular output action provides a quantitative indication of the criticality of the output action. (The benefit loss function can also be viewed as a generalization of the time-value function proposed in [Jen85].)

When the benefit losses accumulated over a certain period of time exceed a certain predetermined threshold, we can treat the RT computer system as having failed in its application mission. The threshold can thus be called a system failure threshold and it should be determined by the system engineer.

Given the specifications of expected component failures (or various fault sources) as probabilistic variables, the expected benefit loss caused by the fault sources is also a probabilistic variable. If the system engineer provides the specification of the benefit loss function in addition to the expected component failure specification, then the expected benefit loss can be calculated. Or if only frequency bounds on component failures are provided without information on probability distributions, then just the maximum expected benefit loss can be calculated. Then in principle, a choice between two or more options for degrading selected service functions due to the resource shortage can be made on the basis of expected benefit loss values under each optional configuration.

### 2.3 Cost and maintainability requirements

The cost constraints that are always present in any system development situation consist of both development cost constraints and operating cost constraints. The development cost includes the following items:
(1) Common off-the-shelf (COTS) hardware and software components;
(2) Design, manufacturing, and integration of new hardware and software components.

The operating cost may include the following items:
(1) Human operator effort;
(2) Power consumption, space occupancy, and weight in some application environments.

The maintainability and expandability requirements can be viewed as the *constraints on the costs for making changes in the system capabilities*. Therefore, they are in a sense a part of the cost constraints.

### 2.4 Main optimization problem

Based on the engineering philosophy and principles laid out in the preceding three sections, we can now formulate the design optimization problem as follows.

**<< Idealistic Optimization Problem Statement >>**

*OPTIMIZE THE TRADE-OFF BETWEEN*
- Total expected benefit loss
- Cost (for new development, maintenance, and operation)

*GIVEN*
- Expected behavior of design-time fault sources
- Expected behavior of run-time fault sources.

*VARIABLES*
- COTS hardware and software components
- custom designed hardware and software components
- System architecture, design, and synthesis
- Human operator responsibility.

Here the total expected benefit loss is expressed as:

(1) $\Sigma_{Vi,\ i=1,2,---,}$ (BL(Vi) * Pr[loss of Vi]), where each Vi, i=1,2,---, denotes a service function responsible for a sequence of service output actions,
(2) $\Sigma_{Vi,\ i=1,2,---,}$ [ $\Sigma$ {BL(expected accuracy loss in a combination of output actions $v_i$-comb) | $v_i$-comb is any subset of the output actions expected from the service function Vi} ], or
(3) a combination of (1) and (2).
One implication of the expression (2) is that the benefit loss due to accuracy losses in a combination of output actions, say $\Delta_{vi,1}$, $\Delta_{vi,2}$, and $\Delta_{vi,3}$, is not necessarily equivalent to the sum $BL(\Delta_{vi,1}) + BL(\Delta_{vi,2}) + BL(\Delta_{vi,3})$. However, in practice, only a simple benefit loss function can be provided by the system engineer. For example, the system engineer may say that loss of two consecutive output actions from a service function causes the same benefit loss as the permanent loss of the service function does.

Similarly, the expected behavior of the fault sources can be represented only in simple forms in practice. To be exact, expressions for expected occurrences of observable fault symptoms, e.g., fault rate of a processor, a memory module, etc., are more practical substitutes for the expressions for the fault source behavior. Even the characterization of the observable fault symptoms is not fully within the state of the art yet.

The items listed under the heading "Variable" in the idealistic problem formulation are those which the system engineer can change to reach the optimization goals.

A large number of variations of the idealistic optimization problem statement are conceivable and may be posed in different applications. Three examples are given below.

**<< Variation I >>**

*MINIMIZE* Total expected benefit loss
*SUBJECT TO* Cost < $X

*GIVEN*
- Fault-rate(processor)
- -----

*VARIABLES*
- -----

**<< Variation II >>**

*MINIMIZE* Cost
*SUBJECT TO* Total expected benefit loss < X

*GIVEN*
- Fault-rate(processor)
- -----

*VARIABLES*
- -----

**<< Variation III >>**

*MAXIMIZE* Tolerable Fault-rate(processor)
*SUBJECT TO* Total expected benefit loss < X
Cost < $Y

*GIVEN*
- -----

*VARIABLES*
- -----

## 3. Challenges in resource allocation

One useful property of the optimal design methodology framework presented in Section 2 is the illumination of the separation between the design activities that are amenable to automation and those activities which must be handled by the system engineer. In a sense, the methodology reduces the latter activities to a minimum. Nevertheless, those activities pose great challenges to the system engineer trying to realize truly optimal resource allocation. In this section, those activities are discussed along with other challenges posed to the researchers aspiring to establish a systematic methodology for producing an optimal design with the given specification of the RT functional requirements, types of faults to be tolerated, and the benefit loss function.

### 3.1 Dependence on subjective judgments by the system engineer

#### 3.1.1 Determination of the benefit loss function

Consider a command and control ($C^2$) system which examines a segment of the sky, detects an unfriendly reentry vehicle (RV), and commands an interceptor to destroy the RV. The $C^2$ system performs the following two service functions among others:
(1) Controlling a radar and
(2) Ordering interceptors to act at appropriate times.

Assume that the high-level defense algorithm adopted is to intercept an RV when the altitude of

the RV is within a certain range $[a_1, a_2]$. After issuing an intercept order, the $C^2$ system checks for an acknowledgment from the interceptor and if it does not receive an acknowledgment within Y seconds after issuing an order, it reissues the order. Complete loss of the interception ordering function will lead to the total failure of this $C^2$ system and thus the benefit loss due to such function loss should be defined as the maximum, say 1000, in the value range adopted [0, 1000]. The system failure threshold can be less than 1000 in such a situation. If an RV is expected to maintain its altitude within $[a_1, a_2]$ only for 10 seconds, absence of an interception order for 10 seconds after the RV reaches the altitude of $a_2$ creates the same damage as the complete loss of the interception ordering function does. An interception order issued after a certain time-point will be useless. Therefore, a (timed value) accuracy requirement on this output action as well as the benefit loss function for this output action can be specified without difficulty.

On the other hand, the case of the radar control service function is different. Complete loss of the radar control function will again lead to the total failure of the $C^2$ system but occasional late issuing (or even omission) of radar orders will not necessarily lead to the total failure of the $C^2$ system. Therefore, while it is easy to determine the benefit loss due to the complete loss of the radar control function, it is not so easy to determine the benefit loss due to accuracy loss in an individual radar ordering action. An example of a practical choice for the system engineer is to define the benefit loss due to a single omission of a radar order for tracking an RV as a small number, say 50, and the benefit loss due to the third consecutive omission of a radar order as a number close to the system failure threshold, meaning a nearly catastrophic event. Clearly, the system engineer must consider many factors in determining these benefit loss functions.

If a processor shortage situation develops due to substantial processor losses, then the $C^2$ system can reduce the processor time allocated to the radar control function by 1/3 since skipping every third control cycle will cause relatively moderate benefit losses.

To the authors' knowledge, specification of the benefit loss function is not a common practice yet. It appears to be a highly meaningful topic for experimental research.

### 3.1.2 Causal relationship between fault sources and observable fault symptoms

System faults (or component failures) are caused by run-time fault sources such as physical damage, material deterioration, electromagnetic noise, etc., and/or design faults. Run-time fault sources are basically physical phenomena and thus a body of knowledge exists on how to characterize their behavior. On the other hand, quantitative characterization of the causal relationship between their behavior and observable fault symptoms (occurring with the RT computer system) is an inherently difficult problem. Therefore, in practice, the system engineer must produce models (expected occurrence patterns) of observable fault symptoms based on statistical measurement data available. Although considerable research efforts have been invested to obtain such statistical data, the available database appears insufficient (or at least not well utilized by system engineers for various reasons).

### 3.1.3 Cost characterization

The cost factors to be considered in the optimal design process include in general:
(1) design cost,
(2) maintainability and expandability, i.e., costs of handling changes,
(3) material cost,
(4) power, weight, volume, and
(5) operator cost.
Items (1), (2), and (5) are not amenable to simple quantitative treatment. It is inevitable to rely on some heuristic judgment by the system engineer who has access to some experience data.

## 3.2 Other challenges

### 3.2.1 Relationship between observable fault symptoms and inaccurate system outputs

Analyzing how a specific observable fault symptom leads to accuracy loss in output actions has been a subject of active research for at least two decades. Such a causal relationship is of course a function of the system architecture, in particular, the types of fault tolerance mechanisms incorporated and the way the mechanisms are embedded. A number of RT fault tolerance techniques of fundamental nature have been established [Kim94]. However, there is still need for development of cost-effective integrations of basic RT fault tolerance techniques. Therefore, some crude forms of systematic analyses of the relationship between

observable fault symptoms and inaccurate system outputs are within the state of the art but the deeper study of the relationship still remains to be an urgent research topic.

### 3.2.2 Multi-step resource allocation

The idealistic optimal design problem statement given in Section 2.4 defines a very large design space consisting of many variables. Due to the unmanageable complexity involved in finding truly optimal points in such a large space, resource allocation occurs in multiple steps in all practical system development, thereby sacrificing the optimality somewhat for drastically reduced complexity in resource allocation. In this process, the system design is accomplished in a hierarchical form and a manageable instance of the idealistic optimization problem statement can be used in each step. In general, the following sequence of five steps occurs.

**(Step 1) Selection of sensors, actuators, and the control strategy.**

The components selected in this very first selection step are related in the following manner: "if these sensors operate according to their specifications, the control computer system will process sensor data according to the control strategy adopted and as a result, timed control values (i.e., actuator commands) will be sent to the actuators which, if operate according to their specifications, will impact the application environment in ways consistent with the control strategy. "
The specification of each sensor which should be formulated before or during this selection step must include a specification of the timed value output requirements. These outputs go to the control computer system. Similarly, a specification of the timed value output requirements associated with the actuators selected must be formulated before or during this selection step. The outputs of the actuators go to the application environment. The specification of the timed value output requirements for the control computer system is produced as a part of this first selection step.

The design of fault tolerance and reconfiguration capabilities at this level must reflect the cost-effectiveness of using redundant sensing schemes, redundant actuation schemes, or redundant control computer system architectures. These design choices have relationship to both static resource allocation and dynamic allocation.

**(Step 2) Determination of distributed computing subsystems partly based on sensor and actuator locations.**

Due to the geographically dispersed nature of the sensors and actuators and for other reasons, the system engineer may choose to use a wide-area distributed system structure in implementing the control computer system. If the distributed system approach is adopted, then the system engineer produces specifications of various distributed computing subsystems and their communication structures, which include the specification of the timed value output requirements for each subsystem as well as the specification for each communication structure. Obviously adopting such an architecture means some decisions on resource allocation.

The design of fault tolerance and reconfiguration capabilities at this level must reflect the cost-effectiveness of using redundant subsystems.

**(Step 3) Determination of a LAN based distributed architecture for each subsystem in an wide-area system .**

The system engineer may choose to implement a subsystem in an wide-area distributed computing system as a local area network (LAN) based subsystem instead of making it a single cabinet subsystem. Such a choice also means some decisions on resource allocation. If the LAN based approach is adopted, the system engineer must produce the specification of the timed value output requirements for each cabinet (or node) in the LAN based subsystem.

The design of fault tolerance and reconfiguration capabilities at this level must reflect the cost-effectiveness of using redundant cabinets (or nodes).

**(Step 4) Determination of a parallel machine architecture for each cabinet (node) in a LAN based subsystem and associated application function partitioning.**

For implementation of each cabinet (node) in a LAN based subsystem, the system engineer may choose to adopt a parallel machine architecture instead of a conventional uniprocessor or dual-processor machine architecture. If a parallel machine architecture is adopted, the system engineer must then decide whether to partition both the set of processors in the parallel machine and the set of

application functions into processor-groups and function-groups, respectively, such that a processor-group-to-function-group mapping is defined. All these are obviously instances of static resource allocation decisions.

The design of fault tolerance and reconfiguration capabilities at this level must reflect the cost-effectiveness of using redundant processor-groups (possibly coupled with function-groups).

**(Step 5) Fine-grain time multiplexing of the hardware in a processor-group for multiple application functions**

This final step is the subject which has been the most extensively addressed in the research field of computer resource allocation so far. Many time-sliced round-robin scheduling, fixed priority scheduling, deadline-driven scheduling, and other simple-structure task scheduling approaches have been produced from such research. The allocation here is of the dynamic allocation type. Most task/process models used in previous research do not reflect the application semantics much. We feel that this must change in order to advance the state of the art in this area in a substantial way.

The design of fault tolerance and reconfiguration capabilities at this level must reflect the cost-effectiveness of using redundant processors (possibly coupled with application functions).

**Acknowledgment**: The work by the first co-author, Kane Kim, was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the University of California MICRO Program under Grant No. 93-080, and in part by USAF Rome Laboratory and Harris Corporation under Contract No. 0277-3003483 (a subcontract of Contract F30602-91-D-0042/D0-0033, performed during June - Dec. 1993).

## References

[Hal94] Halang, W.A. and Stoyenko, A.D. eds., 'Real Time Computing', NATO ASI series F, Vol. 127, Springer-Verlag, 1994 (Proceedings of the NATO Advanced Study Institute on Real Time Computing held in Sint Maarten, Oct. 1992).

[How93] Howell, S., Hoang, N., Nguyen, C., and Karangelen, N., "Critical Issues in the Design of Large-Scale Distributed Systems", Proc. IEEE Workshop on Advances in Parallel and Distributed Systems, Oct. 1993, Princeton, pp. 28-33.

[Jen85] Jensen, E.D., Locke, C.D., and Tokuda, H., "A Time-Value Driven Scheduling Model for Real-Time Operating Systems", Proc. IEEE CS Symp. on Real-Time Systems, Nov. 1985.

[Kim92] Kim, K.H. and Lawrence, T.F., "Adaptive Fault Tolerance in Complex Real-Time Distributed Computer System Applications", Computer Communications, Vol. 15, No. 4, May 1992, pp.243-251.

[Kim94] Kim, K.H., "Action-Level Fault Tolerance", Chapter 17 in S.H. Son ed., 'Real-Time Systems', to be published by Prentice-Hall in 1994.

[Ran75] Randell, B., "System Structure for Software Fault Tolerance", IEEE Trans. on Software Engineering, June 1975, pp.220-232.

# INTEGRATED COMPLEX SYSTEM ENGINEERING METHODOLOGY AND TOOLSET

Ed P. Andert Jr.
Conceptual Systems & Software
P.O. Box 727, Yorba Linda, CA 92686
andert@oac.uci.edu

Lawrence Peters
Software Consultants International Ltd.
P.O. Box 5712, Kent, WA 98031

This paper describes an integrated methodology for the overall process of engineering complex systems. The approach addresses the fundamental areas where current support methods and tools are lacking which leads to the extended length and expense of today's complex system developments. The methodology builds on existing complex system design, MOE/ requirements specification, assessment, and distributed system analysis technologies. Emerging standards, alternative design view and comprehensive system evaluation methodologies are integral parts of the approach.

## Improving System Development

Today's large-scale systems respond in complex ways to external conditions. Such systems are typically implemented on parallel, distributed and heterogeneous architectures. Development is lengthy and expensive.

System engineers need to evaluate the behavior of large-scale system's throughout their development. However, methods and techniques for system design and evaluation need improvement in several fundamental areas to allow developing large-scale systems [Choi 92]. The areas are discussed as follows.

### *Behavior Evaluation at Different Stages*

For large-size, complex developments, system engineers need to evaluate behavior at different stages of design. In many large-scale developments evaluation is left until system testing. This results in a design and implementation where performance is often inconsistent with client needs. Revisions that must be made late in the development life cycle are expensive.

### *Fragmented Design and Evaluation Methods*

Various system design and evaluation methods are fragmented because they were developed independently. This fragmentation makes it extremely difficult for engineers to identify critical functions of a design before the system is physically built and tested. Non-integrated simulation typically provides limited critical function identification. The diversity of system representations makes the integration of existing methods unachievable.

### *An Understandable Methodology is Needed*

An understandable methodology is needed that integrates complex system design and evaluation across system aspects and design stages. It should allow evaluation of control, functional and resource/data behaviors.

System behavior needs to be addressed in a hierarchical manner to accommodate different levels of complexity and fidelity. System engineers, domain experts, program managers and other interested parties should be able to easily understand system behavior (at different levels of complexity). Methodologies need the ability to represent and/or give access to informational, functional, behavioral, implementation and environmental views of a design.

The system design and evaluation methodology also needs a robust representation that allows direct linkage between design and evaluation. System evaluation and suggested

improvements need to be based on requirements, design and experience.

### *Addressing Distributed & Real-time Concerns*

Methodologies for system development and evaluation need to address large-scale and real-time concerns. The control states and response times need to be carefully measured and guaranteed.

High speed distributed and parallel hardware is increasingly necessary to meet the real-time needs of advanced military systems. However, rarely is the full capacity of such powerful hardware realized. In many cases only a small fraction of the potential power can be effectively utilized. Poor performance is often caused by:

- The application does not take advantage of available concurrent resources.
- The application is inefficiently partitioned and mapped onto processors.

The complexity of concurrency issues is far beyond the capabilities of today's development tools. As a result, development for distributed systems requires much optimization effort. Thus, a system engineering methodology should allow the design, efficient partitioning and performance evaluation of parallel/distributed systems.

## Integrated Complex System Engineering

This paper discusses a methodology and toolset concept to address the needs raised in the previous section. It provides an integrated means of engineering and analyzing large-scale complex systems.

The integrated engineering approach addresses the deficiencies of large-scale system support methods as follows:

- Evaluation of system behavior at different design stages is facilitated. The methodology allows varying design views and hierarchical system design. Integrated evaluation ranges from requirements to detailed design.
- Evaluation is tightly integrated with design. System engineers can focus on different system aspects or design stages. Formal and informal methods complement one another. Evaluation can be performed prior to system build and test. Requirements definition is integrated facilitating analysis of compliance, conflict, and critical area identification.
- The methodology is robust and understandable. Design representation is hierarchical and multifaceted to accommodate complexity and fidelity. The design representation and evaluation feedback maximizes understandability by engineers and domain experts. In addition, translation and viewing of mainstream tool standards is integrated to facilitate maximum understanding and reengineering.
- Parallel, distributed, real-time and other complex system concerns are addressed. Support for efficient use of parallel and distributed systems is provided. Control states and response times can be carefully specified, designed and measured.

The methodology and subsequent toolset concentrate on integrating the following capabilities into a single environment:

☆ *Measures of Effectiveness (MOE)/ Requirements specification* in an English-like form using advanced graphical user interface.

☆ Hierarchical *system design definition* that supports alternative design views to improve understandability.

☆ Integration with common and emerging design and evaluation *tool interchange standards* (CDIF, ECBS, DESTINATION, EIA) for understandability and reengineering.

☆ *Support for developing parallel and distributed applications* including resource analysis, trade-off and efficient allocation of software to resources.

☆ *System design simulation* capability.

☆ *Comprehensive design evaluation* including critical area, performance, and problem analysis using modeling and expert system technologies.

The methodology addresses Navy ECS areas of: design capture views; MOE and requirements definition and evaluation; design optimization; evaluation; assessment; reengineering; and integration. An important feature of this approach is the ability to provide developers with *suggested improvements* for system design problems. The evaluation methodology identifies critical design areas as early as possible in the development lifecycle.

The integrated complex system engineering approach builds on three different advanced technologies. These technologies are enhanced and integrated into a cohesive complex system development and evaluation environment as shown in Figure 1. The basis technologies are:

- ✓Intelligent Real-Time System Assessment Tool (ExpeR/T) sponsored by NSWC,
- ✓ProTEM™ system design and simulation methodology and tool (original development sponsored by NSWC),
- ✓Parallel Software Visualization and Automated Partitioning (ParVAP) tool sponsored by ARPA.

### *Integration with Interchange Standards*

The methodology concept includes translation techniques for tight integration with emerging interchange standards. The engineering and evaluation tools are tightly integrated into a cohesive unit with advanced import/export capability. The purpose of the advanced translation technology is to allow access to system capture information from external commercial tools such as Teamwork, RDD-100, Statemate, etc. Access to external tools is needed for reengineering and usage of other representations when they are more appropriate.

This advanced translation capability will go beyond the CDIF interchange model to address emerging standards such as ECBS (SETIS) [White 93] and DESTINATION [Computer 92]. The tool will be a model multifaceted environment that encourages the use of other design/evaluation tool results.

### *Alternative Design Views*

The alternative design view approach enhances the core system modeling and simulation methodology with both hierarchical and alternative views. The core system modeling methodology is Petri net based.

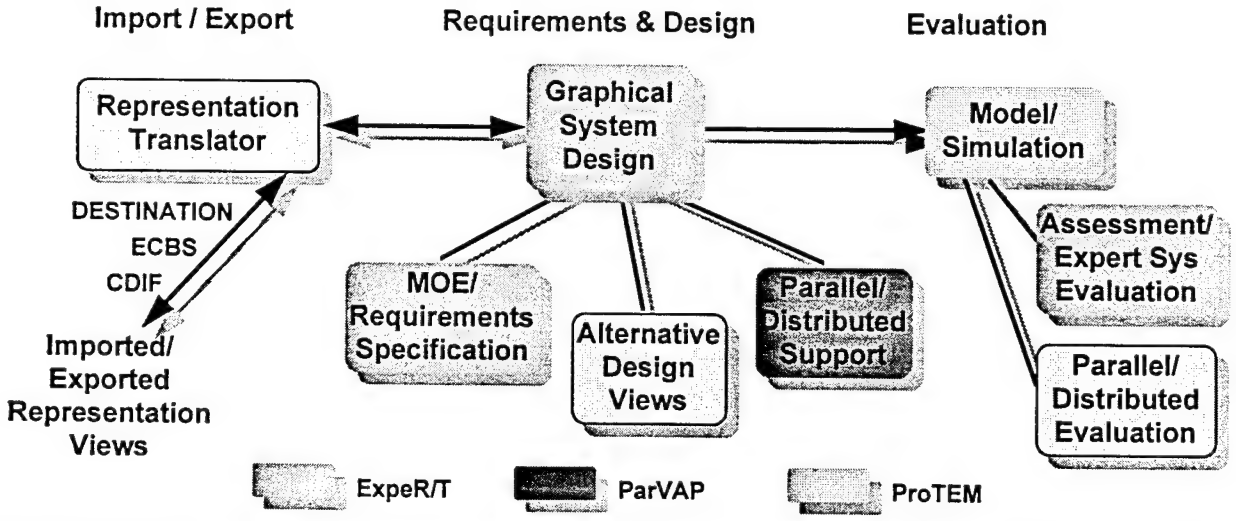



*Figure 1. Integrated complex system engineering approach builds on and enhances three advanced technologies.*

The basic methodology for hierarchical Petri nets is detailed in [Zaidi 91]. Essentially, the lowest level in the hierarchy is a detailed Petri net. Higher levels in the hierarchy detail relationships between lower level Petri-nets.

Our approach for alternative views is to use icon-based diagrams coupled with existing technology. Developers can select from a library of system component icons which are linked in conformance with one of the following views:

- *Informational* view represents the system in abstract terms with component icons linked by entity-relationships and having attribute style descriptions,
- *Implementation* view defines the physical resources with interconnected component icons that have descriptions, parameters and characteristics,
- *Behavioral* view is captured with hierarchical ProTEM Petri nets,
- *Functional* view defines functional decomposition using integrated data flow diagram and data dictionary description,
- *Environmental* view defines system constraints, conditions, and measurement through text-like hierarchical MOE and requirement specification (ExpeR/T).

System component icons are associated with Petri net elements for tight linking to the formal, robust Petri net representation and modeling.

### *Comprehensive Evaluation and Assessment*

The approach for comprehensive evaluation and assessment builds on system assessment in the ExpeR/T methodology. This approach integrates and enhances the base technology with the cability to evaluate parallel / distributed systems. A combined algorithmic and expert system approach is used to assure developers of the performance, redundancy, fault tolerance, etc. characteristics of a distributed system definition.

### *Complex System Design and Modeling*

The complex system specification, modeling and assessment capability of this methodology is based on the results of two NSWC SBIR projects.

The first SBIR project basis is an Intelligent Real-Time System Assessment Tool (ExpeR/T). The ExpeR/T methodology concentrates on allowing the definition of MOEs and requirements to perform critical component evaluation and problem analysis [Andert 94, 93a]

The second basis for the methodology is also a component of the ExpeR/T tool [Andert 93b]. This product, ProTEM™, is a system design and simulation methodology and tool, developed under a SBIR Phase II project sponsored by NSWC. It is currently a commercial software product.

#### *MOE and Requirements Specification*

The requirements specification approach allows selection of "system factors" from a menu. Selecting a factor from the menu leads to a sub-factors list allowing further selection, etc. Details for a sub-factor can be defined using a "template". The taxonomy of factors and sub-factors is essentially derived from those defined in [Nguyen 92].

#### *System Design and Modeling*

The ProTEM™ complex system design and simulation tool supports an extremely effective technology for modeling and simulating systems. This technology is based on extended Petri net theory. The extended form of Petri net technology enables users to model all facets of complex systems including sequentiality, concurrency, asynchrony, priorities and timing. This is accomplished through the use of hierarchical network modeling techniques using simple building blocks. Petri net technology has been

identified as a formal method capable of addressing complex real-time system modeling and simulation [Choi 92].

*System Assessment*

The approach for systems assessment is derived from the ExpeR/T tool. Assessment concentrates on system performance analysis, critical component identification, problem analysis, and critical component correction assistance. Heuristic and algorithmic assessment metrics are captured with expert system technology yielding robust and expandable behavior.

### *Parallel and Distributed System Analysis*

Systems engineers typically define the requirements and initial design for complex software systems. System functions are partitioned onto parallel and distributed processors very early in the design process using classical "functional block diagram" or "data flow" techniques. These techniques often do not consider the efficient partitioning of processing onto resources.

We have begun the process of developing a parallel and distributed system development approach to encourage early consideration of efficient resource utilization. The approach, called ParVAP, provides a visualization environment to capture algorithm and data object dimensions that have the potential to be mapped onto concurrent processors. It further provides automated partitioning to produce efficient programs for different parallel architectures and topologies.

## References


[Andert 94] E. Andert and L. Peters, *Computer-aided design assessment tool*, Proc. of IEEE Intl. Conference on Computers and Communications, AZ, April, 1994.

[Andert 93a] E. Andert and L. Peters, *System design metrics through simulation and intelligent assessment*, AIAA Proc: of Computers in Aerospace 9 Conf., CA, Oct. 1993.

[Andert 93b] E. Andert and L. Peters, *An intelligent real-time system assessment tool*, Proc. of the Complex Sys. Eng. Synthesis and Assess. Tech. Wksp, MD, July, 1993.

[Choi 92] D. Choi, J. Youngblood, S. Howell, & P. Hwang, *Systems modeling*, NSWC publication NAVSWC TR 91-592, Dahlgren, VA, Feb. 1992.

[Computer 92] *System engineering automation (SEA) for distributed systems*, Computer Command and Control Company, Final Report, Dept. of the Navy, Contract No. N00014-91-C-0183, Arlington, VA, Aug. 1992.

[Nguyen 92] C. Nguyen and S. Howell, *System design factors*, Proc. of the Complex Sys. Eng. Synthesis and Assess. Tech. Wksp, MD, 1992.

[White 93] S. White, et al., *Systems engineering of computer-based systems*, IEEE Comp., Nov, 1993.

[Zaidi 91] S. Zaidi, *On the generation of multilevel distributed intelligence systems using Petri nets*, George Mason Univ. puglication GMU/ C3I-112-TH, Fairfax, VA, Nov., 1991.

# APPENDIX A

## LIST OF PANELS

ESSENTIAL DOMAIN MODELS PANEL
Chair: Evan Lock
Members: Nick Karangelen, Phil Hwang, Kane Kim

EARLY EVALUATION PANEL
Chair: Jane Liu
Members: Norman Schneidewind, Mac McCoy, Tim Ramsey, Farnam Jahanian, Robert Thompson

NEW PARADIGM PANEL
Chair: Stephanie White
Members: Dennis Buede, Tom Chosinski, Dick Cobb, Mike Cochran

APPENDIX B

LIST OF ATTENDEES

DR. BEN ABBOTT
VANDERBILT
83 TROTWOOD CIR
BRENTWOOD TN 37027-
P: (615)331-9771
F: (615)343-6702
E: abbott@vuse.vanderbilt.edu

MR. AHMAD ABUALSAMID
UNIV OF WI
2401 POST RD #201
MADISON WI 53713-
P: (608)277-7778
F:
E: ahmadz@cae.wisc.edu

DR. VENKATESH AKELLA
UNIV OF CA
M/S: DEPT OF ECE
DAVIS CA 95616-
P: (916)752-9810
F: (916)752-8428
E: akella@ece.ucdavis.edu

MR. CARLOS AMARO
NJIT
M/S: RTCL
15 OGDEN CT
JEFFERSON NJ 07438-
P: (215)208-1123
F: (215)596-6777
E: amaro@earth.njit.edu

MR. THEODORE BAPTY
VANDERBILT
2213 29TH AVE S
NASHVILLE TN 37212-
P: (615)297-3643
F: (615)343-6702
E: bapty@vuse.vanderbilt.edu

MS. SHERRY BARKER
NSWCDD
M/S: CODE B10
DAHLGREN VA 22448-5000
P: (703)663-7378
F: (703)663-1952
E: sbarker@relay.nswc.navy.mil

MR. ERIC BREHM
AST INC
12200 E. BRIARWOOD AVE. STE 26
ENGLEWOOD CO 80112-
P: (303)790-4242
F: (303)790-2816
E: 72760.663@compuserve.com

MR. JACK BRINKER
TRW
M/S: 953/1140
P.O. BOX 1310
SAN BERNARDINO CA 92402-1310
P: (909)382-8494
F: (909)382-2000
E: jack_brinker@pc007.nafb.trw.com

MR. DAVID BRITTON
TRIDENT
10201 LEE HWY STE 300
FAIRFAX VA 22030-
P: (703)691-7792
F: (703)273-6608
E: dbritton@bass.gmu.edu

DR. DENNIS BUEDE
GMU
FAIRFAX VA 22030-
P: (703)993-1727
F: (703)993-1706
E: dbuede@mason.gmu.edu

MR. MICHAEL CASEY
TRIDENT
10201 LEE HWY STE 300
FAIRFAX VA 22030-
P: (703)691-7768
F: (703)273-6608
E: mcasey@gmuvax.gmu.edu

MR. JIN-YOUNG CHOI
CCCC
2300 CHESTNUT ST STE 230
PHILADELPHIA PA 19103-
P: (215)854-0555
F: (215)854-
E: choi@saul.cis.upenn.edu

MR. THOMAS CHOINSKI
NUWCDET
M/S: CODE 2151
BUILDING 80
NEW LONDON CT 06320-
P: (203)440-5391
F: (203)440-5243
E: choinski@ginbox.nl.nusc.navy.mil

MR. CARL CRAWFORD
NSWCPHD
DAM NECK
VIRGINIA BEACH VA 23461-5300
P: (804)433-7584
F: (804)433-6809
E: ccrawford@hercules.nswces.navy.mil

DR. HARRY CRISP
NSWCDD
M/S: CODE B05
DAHLGREN VA 22448-
P: (703)663-1696
F: (703)663-1695
E: hcrisp@relay.nswc.navy.mil

MR. JAMES ECK
DZIS
P.O. BOX 7126
GAITHERSBURG MD 20898-
P: (301)640-4058
F: (301)840-0737
E:

MR. MICHAEL EDWARDS
NSWCDD
M/S: CODE B44
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
P: (301)394-4187
F: (301)394-1175
E: medward@relay.nswc.navy.mil

DR. WILLIAM FARR
NSWCDD
M/S: CODE B10
DAHLGREN VA 22448-5000
P: (703)663-8388
F: (703)663-4568
E: wfarr@relay.nswc.navy.mil

DR. BOUFARES FAOUZI
UNIV PARIS XIII
AVE JP CLEMENT
VILLETANEUSE FRANCE 93470
P: 49.40.31.44
F:
E:

MR. JOHN FINI
STRATEGIC INSIGHT
2011 CRYSTAL DR STE 101
ARLINGTON VA 22202-
P: (703)553-9700
F: (703)553-9665
E:

DR. ARMEN GABRIELIAN
UNIVIEW SYSTEMS
1192 ELENA PRIVADA
MOUNTAIN VIEW CA 94040-
P: (415)968-3476
F: (415)968-3476
E: armen@well.sf.ca.us

MR. MICHAEL GORLICK
THE AEROSPACE CORP
2350 E EL SEGUNDO BLVD
EL SEGUNDO CA 90245-
P: (310)336-8661
F: (310)336-4402
E: gorlick@aero.org

MS. AMANDA HARDEN
SIMMS INDUSTRIES INC
4471 JAMES MADISON PKWY #102
DAHLGREN VA 22448-
P: (703)663-3943
F: (703)663-3505
E:

MR. MATTHEW HARELICK
NJIT
155 SUMMIT ST OAK 800
NEWARK NJ 07103-
P: (201)621-7914
F:
E: matth@earth.njit.edu

MR. STEVE HARRISON
NUWCDET
M/S: CODE 2153
NEW LONDON CT 06320-
P: (203)440-6153
F: (203)440-5987
E: harrison@code20.nl.nuwc.navy.mil

DR. HERBERT HECHT
SOHAR INC
8421 WILSHIRE BLVD STE 201
BEVERLY HILLS CA 90211-
P: (213)653-4717
F: (213)653-3624
E: herb@sohar.com

MR. HENRY HEFFERNAN
EDP NEWS SERVICE
19 EYE ST NW
WASHINGTON DC 20001-
P: (202)336-7208
F: (202)789-1880
E:

MS. NGOCDUNG HOANG
NSWCDD
M/S: CODE B44
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
P: (301)394-4877
F: (301)394-1175
E: nhoang@relay.nswc.navy.mil

MR. STEVEN HOWELL
NSWCDD
M/S: CODE B44
10901 NEW HAMPSHIRE AVE.
SILVER SPRING MD 20903-5640
P: (301)394-3987
F: (301)394-1175
E: showell@relay.nswc.navy.mil

DR. MICHELLE HUGUE
TRIDENT
10201 LEE HWY SUITE 300
FAIRFAX VA 22030-
P: (703)273-1012
F:
E: meesh@nemo.cs.umd.edu

MR. PHILIP HWANG
DMA
M/S: CODE A-10
8613 LEE HWY
FAIRFAX VA 22031-2137
P: (703)285-9222
F: (703)285-9396
E: phil@dmso.dtic.dla.mil

DR. FARNAM JAHANIAN
UNIV OF MI
M/S: DEPT OF EECS
ANN ARBOR MI 48109-2122
P: (313)936-2974
F: (313)763-1503
E: farnam@eecs.umich.edu

MR. RALPH JEFFORDS
NRL
M/S: CODE 5546
4555 OVERLOOK AVE SW
WASHINGTON DC 20375-5000
P: (202)404-8493
F: (202)404-7942
E: jeffords@itd.nrl.navy.mil

MR. MICHAEL JENKINS
NSWCDD
M/S: CODE B44
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
P: (301)394-4139
F: (301)394-1175
E: mjjenki@relay.nswc.navy.mil

MR. NICHOLAS KARANGELEN
TRIDENT
10201 LEE HWY SUITE 300
FAIRFAX VA 22030-
P: (703)273-1012
F: (703)273-6608
E: dbritton@bass.gmu.edu

MR. MITCHEL KARP
K&K SOFTWARE ENGR
RT 1 BOX 12A
FLINT HILL VA 22627-
P: (703)675-3893
F:
E:

MR. JAMES KENNER
SETI
12890 EDWIN DR
NOCKSVILLE VA 22123-
P: (703)594-3362
F: (703)594-3273
E:

MR. JEE-IN KIM
CCCC
2300 CHESTNUT ST STE 230
PHILADELPHIA PA 19103-
P: (215)854-0555
F: (215)854-0665
E: kim@cccc.com

DR. KANE KIM
UNIV OF CA
M/S: DEPT OF E/CE
IRVINE CA 92717-
P: (714)856-5542
F: (714)856-4076
E: kane@ece.uci.edu

DR. GARY KOOB
ONR
M/S: CODE 311
800 N QUNICY ST
ARLINGTON VA 22217-5660
P: (703)696-0872
F: (703)696-2611
E: koob@itd.nrl.navy.mil

DR. LETITIA KORBLY
KORBLY & THOMAS
112 LEE AVE #201
TAKOMA PARK MD 20912-
P: (301)270-8020
F: (301)270-6941
E: letitia.korbly@his.com

MR. NAOUFEL KRAIEM
UNIV PARIS I
M/S: MASI/CRI
17 RUE DE TOLBIAC
PARIS FRANCE 75013
P: 44.24.93.65
F: 45.86.76.66
E: naoufel@masi.ibp.fr

DR. JANE W.S. LIU
UNIV OF IL
M/S: DEPT OF CS
1304 W. SPRINGFIELD AVE
URBANA IL 61801-
P: (217)333-0135
F: (217)333-3501
E: janeliu@cs.uiuc.edu

MR. EVAN LOCK
CCCC
2300 CHESTNUT ST STE 230
PHILADELPHIA PA 19103-
P: (215)854-0555
F: (215)854-0665
E: lock@cccc.com

MR. BUFORD LOGAN
ROCKWELL
M/S: FB26
12214 LAKEWOOD BLVD
DOWNEY CA 90241-7009
P: (310)922-5608
F: (310)922-0472
E:

MR. JOSEPH MALEY
BOOZ ALLEN HAMILTON
891 ELKRIDGE LANDING RD STE 15
LINTHICUM MD 21090-
P: (410)684-6483
F: (410)684-3853
E:

DR. THOMAS MARLOWE
SETON HALL UNIV
M/S: DEPT OF M/CS
SOUTH ORANGE NJ 07079-
P: (201)761-9784
F: (201)596-5777
E: marlowe@cs.rutgers.edu

MR. CHRISTOPHER MARTIN
SYSCON
RTE 206, P.O. BOX 1480
DAHLGREN VA 22448-
P: (703)663-9634
F: (703)663-9625
E: chmarti@unode2.nswc.navy.mil

MR. WILLIAM MCCOY
NSWCDD
M/S: CODE B10
DAHLGREN VA 22448-5000
P: (703)663-8367
F: (703)663-4568
E: wmccoy@relay.nswc.navy.mil

MS. CATHERINE MEADOWS
NRL
M/S: CODE 5543
4555 OVERLOOK AVE SW
WASHINGTON DC DC 20375-
P: (202)767-3490
F: (202)404-7942
E: meadows@itd.nrl.navy.mil

MR. WILLIAM MILLER
AT&T BELL LABS
M/S: 15H-408
67 WHIPPANY RD P.O. BOX 903
WHIPPANY NJ 07981-0903
P: (201)386-5339
F: (201)386-6923
E: william.d.miller@att.com

MR. ANDREW MITTURA
SYSCON
RTE 206, P.O. BOX 1480
DAHLGREN VA 22448-
P: (703)663-9653
F: (703)663-9625
E:

MR. GILBERT MYERS
NRAD
M/S: BLDG606/RM230
53140 SYSTEMS ST
SAN DIEGO CA 92152-7560
P: (619)553-4135
F: (619)553-3931
E: gmyers@lyra.nosc.mil

MR. JOHN NALLON
TD TECHNOLOGIES INC
2425 N CENTRAL EXPWY STE 200
RICHARDSON TX 75080-
P: (214)669-9937
F: (214)669-9938
E:

DR. RICHARD NANCE
VPISU
M/S: SRC
320 FEMOYER HALL
BLACKSBURG VA 24061-0251
P: (703)231-6144
F: (703)231-4330
E: nance@vtopus.cs.vt.edu

MR. CUONG NGUYEN
NSWCDD
M/S: CODE B44
10901 NEW HAMPSHIRE AVE.
SILVER SPRING MD 20903-5640
P: (301)394-4550
F: (301)394-1175
E: cnguyen@relay.nswc.navy.mil

MR. DANIEL ORGAN
NUWCDET
M/S: CODE 2151
BUILDING 80
NEW LONDON CT 06320-
P: (203)440-6546
F: (203)440-5243
E: organ@soleil.nl.nuwc.navy.mil

DR. JAMES PALMER
GMU
M/S: ST II RM19
4400 UNIVERSITY DR
FAIRFAX VA 22030-4444
P: (703)993-1507
F: (703)993-1521
E: jpalmer@gmu.edu

MS. LORI PAYNE
SYSCON
RTE 206 P.O. BOX 1480
DAHLGREN VA 22448-
P: (703)663-9778
F: (703)663-9625
E:

DR. DAR-TZEN PENG
ALLIED-SIGNAL
9140 OLD ANNAPOLIS RD
COLUMBIA MD 21045-1998
P: (301)964-4195
F: (301)992-5813
E: dtp@batc.allied.com

MR. PHILIP PLANTE
VITRO CORP
45 W GUDE DR
ROCKVILLE MD 20850-1160
P: (301)231-3473
F: (301)231-1233
E:

DR. BEN POUIBABAI
MSCA INC
211 N UNION ST STE 100
ALEXANDRIA VA 22314-
P: (703)634-4853
F: (703)519-8947
E: 74543.2675@compuserve.com

MR. JANIS PUKITE
DAINA
4111 CENTRAL AVE NE STE 212
COLUMBIA HTS MN 55421-2953
P: (612)781-7600
F: (612)781-7600
E: pukite@daina.com

DR. PARMESH RAMANATHAN
UNIV OF WI
M/S: DEPT OF E/CE
1415 JOHNSON DR
MADISON WI 53706-1691
P: (608)263-0557
F: (608)265-4623
E: parmesh@ece.wisc.edu

DR. BALA RAMESH
NPS
M/S: CODE AS/RA
MONTEREY CA 93943-
P: (408)656-2439
F: (408)656-3407
E: ramesh@nps.navy.mil

MR. TIMOTHY RAMSEY
NICHOLS
4040 S MEMORIAL PKWY
HUNTSVILLE AL 35802-
P: (205)883-1170
F: (205)880-7880
E:

MR. RANDALL RICHARDS
LORAL
9500 GODWIN DR
MANASSAS VA 22110-
P: (703)367-4797
F: (703)367-9440
E: rrichards@lfs.loral.com

MS. NANCY RUNDLET
ZYCAD CORP
100 ENTERPRISE DR STE 500
ROCKAWAY NJ 07866-
P: (201)989-2934
F: (201)989-2940
E: nancy_rundlet@protocol.zycad.com

DR. RICHARD SCALZO
NSWCDD
M/S: CODE A10
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
P: (301)394-2926
F: (301)394-1164
E: rscalzo@relay.nswc.navy.mil

DR. NORMAN SCHNEIDEWIND
NPS
M/S: CODE SM/SS
MONTEREY CA 93943-
P: (408)656-2719
F: (408)656-3407
E: schneidewind@nps.navy.mil

MR. PURNENDU SINHA
NJIT
M/S: DEPT OF CS
UNIVERSITY HGTS
NEWARK NJ 07102-
P: (201)596-2861
F: (201)596-5777
E: pxs3413@hertz.njit.edu

DR. SANG SON
UNIV OF VA
M/S: DEPT OF CS
THORNTON HALL
CHARLOTTESVILLE VA 22903-
P: (804)982-2205
F: (804)982-2214
E: son@virginia.edu

DR. DAVID STEWART
CMU
5000 FORBES AVE
PITTSBURG PA 15213-
P: (412)268-7120
F: (412)268-3890
E: dstewart@cmu.edu

DR. ALEXANDER STOYENKO
NJIT
M/S: DEPT OF C/IS
UNIVERSITY HTS
NEWARK NJ 07102-
P: (201)596-5765
F: (201)596-5777
E: alex@vulcan.njit.edu

MR. MICHAEL TALBERT
VIRGINIA TECH
M/S: SRC
555 MCBRYDE
BLACKSBURG VA 24061-
P: (703)231-7371
F:
E: talbertm@csgrad.cs.vt.edu

MR. JONAH THOMAS
KORBLY & THOMAS
112 LEE AVE #201
TAKOMA PARK MD 20912-
P: (301)270-8020
F: (301)270-6941
E: jethomas@genie.geis.com

DR. ROBERT THOMAS
EG&G WASC
16156 DAHLGREN RD
DAHLGREN VA 22448-
P: (703)663-9373
F: (703)663-0332
E:

MR. ROBERT THOMPSON
TECOLOTE
54 MIDDLESEX TPKE
BEDFORD MA 01730-
P: (617)275-3014
F: (617)275-3407
E:

MR. DINESH VERMA
VPISU
146 WHITTEMORE HALL ISE
BLACKSBURG VA 24061-
P: (703)951-4605
F: (703)231-3322
E: verma@vtvm1.cc.vt.edu

DR. LONNIE WELCH
NJIT
M/S: DEPT OF C/IS
UNIVERSITY HTS
NEWARK NJ 07102-
P: (201)596-5683
F: (201)596-5777
E: welch@vienna.njit.edu

DR. STEPHANIE WHITE
GRUMMAN
M/S: MS B38-35
BETHPAGE NY 11714-3580
P: (516)575-2201
F: (516)575-7528
E: steph@gdstech.grumman.com

MR. MARK WILSON
NSWCDD
M/S: CODE B44A
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
P: (301)394-5099
F: (301)394-1175
E: mlwilso@relay.nswc.navy.mil

MR. GLENN WOOD
TRIDENT
10201 LEE HWY STE 300
FAIRFAX VA 22030-
P: (703)691-7767
F: (703)273-6608
E: trident@digex.com

MR. DAVID ZALEWSKI
NJIT
NEWARK NJ 07102-
P: (201)596-2993
F: (201)596-8365
E: dwz2139@hertz.njit.edu

DISTRIBUTION

| | Copies |
|---|---|
| DOD ACTIVITIES (CONUS) | |
| DEFENSE TECHNICAL<br>INFORMATION CENTER<br>CAMERON STATION<br>ALEXANDRIA VA 22304-6145 | 12 |
| ATTN CODE 5543<br>(KATHERINE MEADOWS)<br>NAVAL RESEARCH LABORATORY<br>4555 OVERLOOK AVE SW<br>WASHINGTON DC 20375 | 1 |
| ATTN CODE 4411B<br>(ELIZABETH WALD)<br>OFFICE OF NAVAL RESEARCH<br>800 NORTH QUINCY STREET<br>ARLINGTON VA 22217-5000 | 1 |
| ATTN CODE E29L<br>COASTAL SYSTEMS STATION<br>DAHLGREN DIVISION<br>NAVAL SURFACE WARFARE CENTER<br>6703 WEST HIGHWAY 98<br>PANAMA CITY FL 32407-7001 | 1 |
| ATTN C3AB<br>(THOMAS LAWRENCE)<br>ROME LABORATORY<br>525 BROOKS ROAD<br>GRIFFISS AFB NY 13441-5700 | 2 |

| | Copies |
|---|---|
| NON-DOD ACTIVITIES | |
| THE CNA CORPORATION<br>PO BOX 16268<br>ALEXANDRIA VA 22302-0268 | 2 |
| ATTN GIFT & EXCHANGE DIVISION<br>LIBRARY OF CONGRESS<br>WASHINGTON DC 20540 | 4 |
| ATTN ED P ANDERT<br>CONCEPTUAL SOFTWARE SYSTEMS<br>P O BOX 727<br>YORBA LINDA CA 92686 | 1 |
| ATTN MAARTEN BOASSON<br>HOLLANDSE<br>SSIGNAALAPPARATEN BV<br>POSTBUS 42<br>7550 GD HENGE LO<br>THE NETHERLANDS | 1 |
| ATTN ERIC BREHM<br>ADVANCED SYSTEM TECHNOLOGIES<br>12200 E BRIARWOOD AVE #260<br>ENGLEWOOD CO 80112 | 1 |
| ATTN RICHARD EVANS<br>GEORGE MASON UNIVERSITY<br>4400 UNIVERSITY DR<br>FAIRFAX VA 22030-4444 | 1 |

DISTRIBUTION (Continued)

| | Copies |
|---|---|
| ATTN VENKATESH IYER<br>PENCOM SOFTWARE<br>9050 CAPITAL OF TEXAS HIGHWAY NORTH<br>AUSTIN TX 78759 | 1 |
| ATTN NAOUFEL KRAIEM<br>LABORATOIRE MASI/CRI<br>UNIVERSITE PARIS I<br>17 RUE DE TOLBIAC<br>F-75013 PARIS<br>FRANCE | 2 |
| ATTN JANE LIU<br>DEPT OF COMPUTER SCIENCE<br>UNIVERSITY OF ILLINOIS<br>1304 WEST SPRINGFIELD AVENUE<br>URBANA IL 61801 | 1 |
| ATTN AZAD MADNI<br>INTELLIGENT SYSTEMS TECH INC<br>3100 DANNYHILL DRIVE<br>LOS ANGELES CA 90064 | 1 |
| ATTN ADRIEN MESKIN<br>ATR<br>15210 DINO DRIVE<br>BURTONSVILLE MD 20866-1172 | 2 |
| ATTN LAWRENCE PETERS<br>SOFTWARE CONSULTANTS INT LTD<br>13812 SE 240TH STREET<br>KENT WASHINGTON 98042 | 1 |
| ATTN LAURA PULLUM<br>QUALITY RESEARCH ASSOCIATES<br>PO BOX 701<br>NORCROSS GA 30091-0701 | 1 |

| | Copies |
|---|---|
| ATTN ALEX QUILICI<br>UNIV OF HAWAII AT MANOA<br>DEPT OF ELECTRICAL ENGINEERING<br>2540 DOLE ST<br>HALL 483<br>HONOLULU HI 96822 | 1 |
| ATTN LONNIE WELCH<br>THE REAL-TIME COMPUTING LAB<br>DEPT OF COMPUTER AND INFORMATION SCIENCE<br>NJIT<br>UNIVERSITY HEIGHTS<br>NEWARK NJ 07102 | 1 |
| INTERNAL | |
| A | 1 |
| A44 (R SCALZO) | 1 |
| B | 1 |
| B02 | 1 |
| B05 (H CRISP) | 5 |
| B10 (S BARKER) | 1 |
| B10 (W FARR) | 1 |
| B10 (W MCCOY) | 1 |
| B10 (D PARKS) | 1 |
| B20 | 1 |
| B30 | 1 |
| B35 (M CHANG) | 1 |
| B35 (R HARRISON) | 1 |
| B35 (M MASTERS) | 1 |
| B42 (J MOSCAR) | 1 |
| B44 | 1 |
| B44 (M EDWARDS) | 1 |
| B44 (N HOANG) | 1 |
| B44 (S HOWELL) | 10 |
| B44 (M JENKINS) | 1 |
| B44 (C NGUYEN) | 1 |
| B44 (H ROTH) | 1 |
| B44 (M WILSON) | 1 |
| B44 (C YEH) | 1 |
| D | 1 |

| | Copies |
|---|---|
| D4 | 1 |
| E231 | 2 |
| E232 | 3 |
| F | 1 |
| G | 1 |
| K | 1 |
| L | 1 |
| N | 1 |
| N24 (R HOLDEN) | 1 |
| N742 (GIDEP) | 1 |

# REPORT DOCUMENTATION PAGE

*Form Approved OMB No. 0704-0188*



| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | 12 January 1995 | Proceedings, 19-20 July 1994 |

**4. TITLE AND SUBTITLE** PROCEEDINGS OF THE 1994 COMPLEX SYSTEMS ENGINEERING SYNTHESIS AND ASSESSMENT TECHNOLOGY WORKSHOP (CSESAW '94), 19-20 JULY 1994

**5. FUNDING NUMBERS**

4B6TBA11A
4B6TBB12A

**6. AUTHOR(S)**

Steven Howell, Coordinator

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Naval Surface Warfare Center Dahlgren Division
White Oak Detachment
10901 New Hampshire Avenue
Silver Spring, MD 20903-5640

**8. PERFORMING ORGANIZATION REPORT NUMBER**

NSWCDD/MP-94/122

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Office of Naval Research
Code 4411B
800 North Quincy Street
Arlington, VA 22217-5000

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

---

**11. SUPPLEMENTARY NOTES**

---

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**

Approved for public release; distribution is unlimited.

**12b. DISTRIBUTION CODE**

---

**13. ABSTRACT** *(Maximum 200 words)*

CSESAW '94 is exploring system level design synthesis and assessment capabilities for large, complex systems. These capabilities will facilitate the development of such systems from informal system requirements, through the design phase prototyping, and into implementation and post deployment. Component products produced by these capabilities are specifications that subenvironments will receive. The focus of the workshop is the development and integration of these multiple technologies and the exploration of the creation of a system level engineering discipline with support technologies to provide potential high payoff solutions to the difficult problems encountered by designers, developers, and maintainers of real-time systems. Further emphasis is on resolving system level technology issues that cut across component boundaries, such as those associated with system behavior requirements of real-time, fault tolerance, cost, and security. Specifically, the theme of this year's workshop is system engineering synthesis for evolutionary systems. Issues addressed within the workshop include requirements specification, requirements traceability, design capture, design evaluation, design optimization, security engineering, and dependability engineering. Panel discussions will emphasize essential domain models for long-life evolutionary systems, new paradigms in system engineering, and early evaluation metrics of system developments.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| CSESAW '94; System Level Design Synthesis and Assessment; System Engineering Synthesis; Optimization, Security Engineering, and Dependability Engineering; Evolutionary Systems; Complex Systems | 269 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | SAR |

## GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and its title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

**Block 1.** Agency Use Only *(Leave blank)*.

**Block 2.** Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

**Block 3.** Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

**Block 4.** Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

**Block 5.** Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

| | | | |
|---|---|---|---|
| **C** | - Contract | **PR** | - Project |
| **G** | - Grant | **TA** | - Task |
| **PE** | - Program Element | **WU** | - Work Unit Accession No. |

**BLOCK 6.** Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

**Block 7.** Performing Organization Name(s) and Address(es). Self-explanatory.

**Block 8.** Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

**Block 9.** Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

**Block 10.** Sponsoring/Monitoring Agency Report Number. *(If Known)*

**Block 11.** Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

**Block 12a.** Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

| | |
|---|---|
| **DOD** | - See DoDD 5230.24, "Distribution Statements on Technical Documents." |
| **DOE** | - See authorities. |
| **NASA** | - See Handbook NHB 2200.2 |
| **NTIS** | - Leave blank. |

**Block 12b.** Distribution Code.

| | |
|---|---|
| **DOD** | - Leave blank. |
| **DOE** | - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports. |
| **NASA** | - Leave blank. |
| **NTIS** | - Leave blank. |

**Block 13.** Abstract. Include a brief ***(Maximum 200 words)*** factual summary of the most significant information contained in the report.

**Block 14.** Subject Terms. Keywords or phrases identifying major subjects in the report.

**Block 15.** Number of Pages. Enter the total number of pages.

**Block 16.** Price Code. Enter appropriate price code ***(NTIS only)***

**Blocks 17.-19.** Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

**Block 20.** Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.